Ingeniería informática

Proyecto final de carrera

Universidad
Carlos III de Madrid

iCanCloud Manager

Gestor de entornos de simulación de cloud computing

Autor: Francisco Javier Paniagua Correa
Tutor: Alberto Núñez Covarrubias
Septiembre de 2011

“El futuro tiene muchos nombres. Para los débiles es lo inalcanzable. Para los temerosos, lo desconocido. Para los valientes es la oportunidad.”

Víctor Hugo

Agradecimientos

A mis padres, por su apoyo incondicional, su cariño y su comprensión, por la paciencia que han tenido y siguen teniendo conmigo. Sin vosotros esto nunca habría sido posible. Gracias por todo.

A mis hermanas, que han sido para mí como mis otras mamás, por su ayuda y porque siempre han estado ahí cuando las he necesitado. Porque aunque en ocasiones estéis lejos siempre os siento muy cerca de mí.

A mis amigos, por estar conmigo en los buenos y en malos momentos y porque siempre me habéis animado y me habéis sacado una sonrisa. A todos los "tronchas" (Pachi, Jose, Carlos, Bere, Víctor,…) por todos los momentos que hemos compartido desde que éramos unos enanos. A todos esos toledanos de la peña "El Fayo" (Marea, Toni, Juan, Luisfer, Julitros, Huevo,…) por esos veranos inolvidables que me habéis hecho pasar.

A mis compañeros de clase, a Alberto González, a Álvaro Menéndez, a Javier Álvarez, a Miguel Ángel Hernández, a Álvaro Villadangos, a Adolfo García y a todos los miembros del "núcleo duro". Gracias por compartir conmigo tantos momentos y por las aventuras que hemos vivido entre estas paredes, y en especial a Sergio Núñez porque ha sacado siempre lo mejor de mí, por ser un gran compañero, confidente y amigo.

A mi tutor, Alberto Núñez, por su tiempo, por su dedicación, por su comprensión y porque siempre me ha ayudado cuando ha habido algún obstáculo en el camino. También a Rosa Filgueira y a Gabriel González, porque vuestra ayuda ha sido muy importante para llegar hasta aquí.

A todos los que habéis compartido este reto conmigo, ya seáis parte del parte o del presente, esto es posible gracias a vosotros. A todos, muchas gracias.

Tabla de contenidos

Índice de tablas

Índice de figuras

Capítulo 1

1. Introducción

El objetivo principal de este proyecto consiste en el desarrollo de una herramienta que gestione entornos de simulación de manera escalable y flexible, que permitan a su vez la experimentación en las nuevas infraestructuras de cloud computing y sus servicios de aplicación. Para ello, dicha herramienta deberá facilitar la interacción entre el usuario y el simulador de computación en nube "iCanCloud".

El concepto de cloud computing (o computación en nube) hace referencia a un modelo que permite ofrecer servicios de computación a través de Internet. Esta nueva tendencia, como se describirá en apartados posteriores, genera beneficios tanto para los proveedores, que pueden ofrecer un mayor número de servicios de forma más rápida y eficiente, como para los usuarios que tienen la posibilidad de acceder a ellos de manera transparente y de pagar únicamente por lo que consumen.

Este proyecto surge en base a una iniciativa para el desarrollo de un simulador encargado de estimar los resultados obtenidos al utilizar una arquitectura de cloud computing para ofrecer una serie de servicios de computación. Así, el simulador *iCanCloud* mencionado anteriormente, surge como consecuencia de la necesidad de realizar pruebas de cloud computing en escenarios donde la infraestructura hardware no es de fácil acceso. En estos casos, la opción de utilizar una solución basada en simulación ofrece ventajas significativas, ya que permite a los clientes del propio cloud probar sus servicios en un entorno controlado, y ajustar el rendimiento resultante antes de pasar a la solución real.

Además, como se mencionó antes, este modelo se caracteriza porque el usuario paga directamente por lo que consume, de modo que la posibilidad de contar con un simulador que estime los costes de la arquitectura en función de los recursos utilizados en la misma se presenta como una opción muy a tener en cuenta.

El número de parámetros configurables que proporciona un simulador como iCanCloud hace que la labor de puesta a punto de una simulación sea compleja y suponga un importante coste de tiempo. De esta manera, la herramienta que se pretende desarrollar tiene como objetivo principal mejorar y facilitar este proceso de configuración de los modelos de cloud, además de otros que se definen a continuación.

1.1 Objetivos del proyecto

El objetivo principal de este proyecto consiste en el desarrollo de una aplicación que gestione entornos de simulación escalables y que interactúe con el simulador *iCanCloud*, todo ello con el objetivo de facilitar a los usuarios de la aplicación la realización de pruebas en escenarios de cloud computing.

Además de este objetivo principal, existen una serie de objetivos secundarios, los cuales son enumeran a continuación:

- **Facilidad de uso**. El sistema surge por la necesidad de facilitar la labor de configuración de las simulaciones, por lo que un objetivo fundamental del mismo debe ser realizar una aplicación que sea lo más intuitiva posible para el usuario y de fácil manejo para el mismo.

- **Flexibilidad de configuración**. El sistema debe permitir definir cada entorno de la forma más flexible posible. De esta forma, el usuario podrá determinar qué elementos forman el cloud que se va a simular y realizar pruebas con distintos valores para así observar la evolución de los resultados obtenidos.

- **Gestión de módulos**. El sistema debe permitir la gestión de un repositorio de los distintos elementos que forman parte de la simulación para de esa forma tener disponible en todo momento la información almacenada en anteriores simulaciones. Así, no será necesario que el usuario se vea obligado a introducir información cada vez que desee lanzar una simulación.

- **Tratamiento de resultados**. El sistema debe recoger la información obtenida como producto de la simulación y tratarla de manera que el usuario pueda tomar conclusiones sobre el trabajo llevado a cabo. Debe mostrar información al usuario acerca de los resultados de la simulación y permitirle exportar los resultados mediante algún tipo de documento (Word, PDF,…)

1.2 Estructura del documento

Este documento se ha dividido en 6 capítulos principales, debido a que se han considerado los oportunos para localizar el proyecto en un contexto y detallarlo en toda su amplitud.

En este primer capítulo se ha comentado brevemente, el marco en el que se encuadra este proyecto y se han descrito cuales son los objetivos que se desean alcanzar con el desarrollo del mismo. En el segundo capítulo, se busca acercar al lector a este marco que comentábamos antes y en el que se encuadra nuestro proyecto: el modelo de "cloud computing". El software en el que se va a apoyar la herramienta a desarrollar es un simulador de clouds. Por ello, es necesario que el lector conozca qué es el cloud computing, cómo surge, qué busca conseguir, etc., y a partir de ahí se verán algunas herramientas de simulación de entornos cloud que hay en el mercado. En el tercer capítulo se realiza el análisis en profundidad del sistema a desarrollar. Para ello se describe la funcionalidad que se desea implementar a partir de los requisitos de información que tiene el usuario de la aplicación. En el cuarto capítulo se realiza el diseño del sistema a partir del análisis obtenido en el punto anterior, describiendo la arquitectura escogida para el sistema, el modelo lógico del sistema con los componentes que lo forman, etc. En el quinto capítulo se mostrará un ejemplo práctico de ejecución de la herramienta y para finalizar, el sexto capítulo incluirá una serie de conclusiones del proyecto y líneas a seguir en el futuro.

Capítulo

2

2. Estado del arte

En este capítulo se va a realizar un estudio del marco en el que se encuadra nuestro sistema, con el objetivo de mejorar la comprensión del desarrollo del mismo. En primer lugar se va a estudiar el modelo de cloud computing en sí (qué es, para qué se utiliza,…), y después se describirán algunas herramientas de simulación que existen en el mercado y que están relacionadas con este modelo de computación.

2.1 Introducción al cloud computing

El papel de las tecnologías de la información cambia rápidamente y actualmente forma una capa invisible que se infiltra poco a poco en todos los aspectos de nuestra vida. Redes eléctricas, control de tráfico, atención médica, suministro de agua, alimentación y energía, además de la mayor parte de las transacciones financieras mundiales, dependen hoy en día de las tecnologías de la información. En 1984 había 1.000 dispositivos conectados a Internet; en 2015 serán 15.000 millones, sometiendo a los sistemas de IT de todo el mundo a exigencias sin precedentes.

Dos modelos de computación continúan dominando las tecnologías de la información: el modelo de ordenador central, de eficacia largamente demostrada, y el más reciente modelo de servidor-cliente. Ahora aparece un tercer modelo - el cloud computing -, creado para responder al explosivo aumento del número de dispositivos conectados a Internet y complementar la presencia cada vez mayor de la tecnología en nuestras vidas y empresas.

El modelo de cloud computing se centra en el usuario y ofrece un modo de adquisición y suministro de servicios muy efectivo. El cloud computing se define y caracteriza por su escalabilidad elástica, por una excepcional experiencia de usuario, y por definir un nuevo modelo económico basado en una nueva forma de consumir servicios.

Cloud computing es una tecnología que consiste en ofrecer computación como servicio. Se trata de una tecnología que permite ofrecer servicios a través de "la nube" de Internet. Todo lo que puede ofrecer un sistema informático se ofrece como servicio en el cloud. De esta forma, podríamos considerar internet como un sistema operativo propio que incorpora software online como servicio para sustituir programas instalados actualmente en nuestros propios ordenadores. Así, en este paradigma tecnológico, la información se almacena de manera permanente en servidores de Internet para enviarse, en el momento en el que se precisa, a cachés temporales de cliente, lo que incluye equipos de escritorio, centros de ocio, portátiles, etc. Esto se debe a que, pese a que las capacidades de los PC han mejorado sustancialmente, gran parte de su potencia se desaprovecha, al ser máquinas de propósito general.

La computación en nube aporta un nuevo enfoque que agrupa y combina las tecnologías ya conocidas, no solo sistemas operativos, sino también bases de datos, servidores, redes de interconexión, middleware, virtualización (abstracción de los recursos de computación), herramientas de gestión, etc. que se han ido desarrollando en las últimas décadas.

Además, el concepto de cloud computing se caracteriza porque es transparente al usuario, de modo que éstos puedan acceder a los servicios disponibles en el cloud sin conocimientos en la gestión de los recursos que usan. Esta característica aparece reflejada de forma simbólica en la Figura 1 que aparece a continuación.

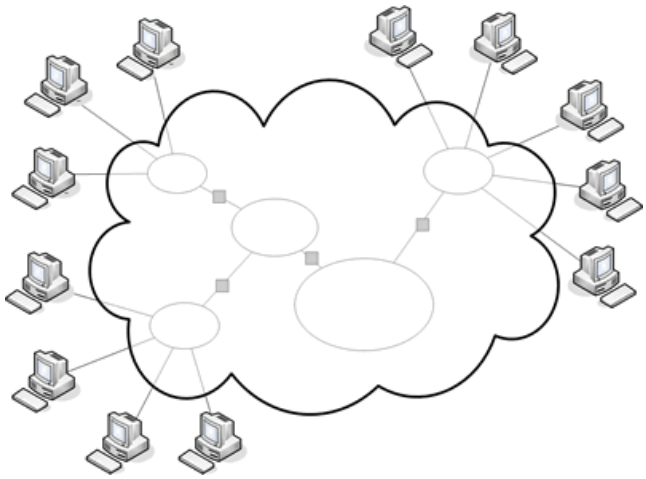

Figura 1: Concepto de cloud computing

De esta manera, todo usuario de la nube podrá acceder a un catálogo de servicios estandarizados y responder a las necesidades de su negocio, de forma flexible y adaptativa, en caso de demandas no previsibles o de picos de trabajo, pagando únicamente por el consumo efectuado.

Además el hecho de que podamos pagar por el uso de los recursos cloud (suscripción) adaptándolos a tus necesidades, permite la optimización de los mismos y redunda en una reducción de los costes finales en comparación con las soluciones tradicionales. Este factor determinante es el que empuja a muchas empresas a cambiar su infraestructura de trabajo y a adquirir soluciones basadas en cloud.

2.1.1 Pago por consumo

Como se comentó anteriormente, existen dos modalidades básicas de pago en cloud computing: el pago por potencial consumo de hardware o software, también denominado suscripción, y el pago por uso.

En el primero se fija un precio fijo por el periodo de computación y se permite hacer uso de la infraestructura las veces que se desee, sin restricciones. Existen diferentes posibilidades: por usuario, en el que se paga en el periodo por el número de usuarios que utilizan la herramienta; por funcionalidad, en el que se paga por el uso de una funcionalidad concreta; y la denominada tarifa plana, en el que se paga una cifra por el periodo, sin restricciones en el número de usuarios ni en los recursos disponibles.

Por su parte, el pago por uso o por consumo, se caracteriza porque se paga por la cantidad consumida de los recursos, normalmente CPU/hora, GB consumidos, ancho de banda de entrada y salida, etc. Esta opción parece la ideal, y es en la que se fundamentan los estudios realizados en el presente proyecto.

Además existe la posibilidad de hacer una mezcla entre las dos modalidades anteriores. Por ejemplo, es común asociar el pago mensual con derecho a un consumo de MB, o número de ciclos de CPU, etc., y en caso de sobrepasar ese consumo pagar la cantidad excedida. Un ejemplo de esto lo encontramos en Zoho Creator.

2.1.2 Elementos ligados al Cloud Computing

La computación en nube se confunde muchas veces con computación en grid (una forma de computación distribuida en la que un super computador virtual está compuesto de un conjunto ó cluster enlazado de ordenadores débilmente acoplados, actuando conjuntamente para realizar tareas muy pesadas), computación autónoma (que puede gestionarse a sí misma) o computación bajo demanda (utility computing). Cloud computing a menudo depende de grids para su implementación, tiene cierta autonomía (por ejemplo, reacciona ante momentos de demanda máxima y reajusta los recursos automáticamente) y se cobra bajo demanda.

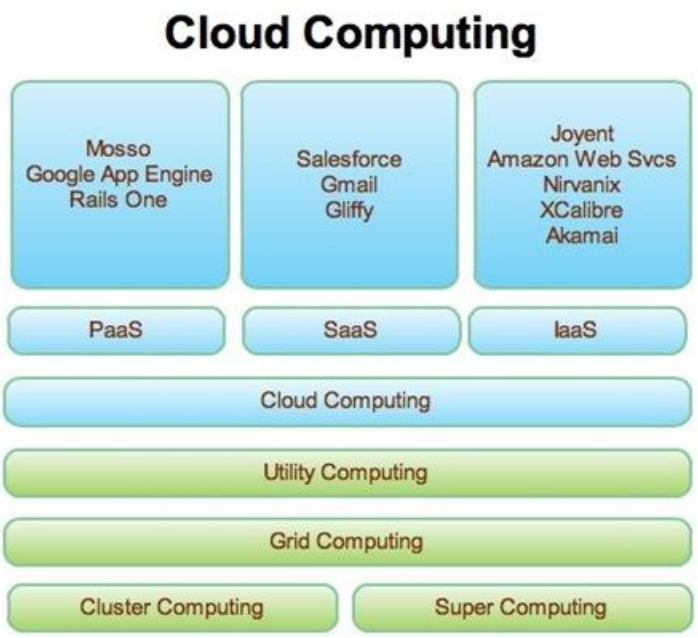

Figura 2: Elementos del cloud computing

Sin embargo, cloud computing tiene un alcance mucho mayor. La esencia de la computación en nube no consiste tanto en las herramientas (software, tecnologías, plataformas, infraestructuras, etc.) que usa, sino en cómo se usan, agrupan e integran, fusionando algunos o muchos de sus principios básicos para crear una dinámica propia y diferenciadora.

La Figura 2 anterior muestra, a modo de resumen, los principales elementos del cloud computing. En azul se muestra a lo que tradicionalmente se refiere como cloud computing, mientras que en verde aparecen algunas de las tecnologías subyacentes que llevaron al desarrollo del mismo.

A continuación se van a describir los modelos en que los suministradores de cloud computing ofertan servicios a sus clientes. Estos conceptos aparecen discretamente en el gráfico anterior y describen más profundamente a continuación. Son los denominados niveles o capas del cloud computing, aunque si pensamos en el concepto de computación como servicio parece más adecuado denominarlos “Modelos de Prestación de Servicio”.

2.1.3 Modelos de Prestación de Servicio

Como decíamos antes, actualmente se pueden distinguir tres modelos de prestación de servicio, que hacen referencia a modelos de negocio de software, entornos y hardware donde las empresas ofrecen el servicio pagado del uso de sus capacidades (Data Centers, Web Service, etc.). Estos modelos son los siguientes:

- **Software como Servicio (SaaS)**. Es el más conocido de los tres modelos y el que suele tener como objetivo al cliente final, aquel que utiliza el software para ayudar, mejorar o cubrir algunos de los procesos de su empresa. Se basa en proporcionar aplicaciones vía Internet. El SaaS es aquella aplicación proporcionada a través de la red, casi siempre a través del navegador o plataformas como Adobe Air, y donde tanto la lógica de la aplicación como los datos residen en la plataforma del proveedor aunque sirven a múltiples organizaciones de clientes (lo que se denomina como multitenencia). El ejemplo de SaaS más ampliamente conocido es Salesforce.com aunque hoy en día existen muchas más, como las Google Apps, las cuáles ofrecen servicios básicos de negocio como el correo electrónico, o Windows Live de Microsoft.

- **Plataforma como Servicio (PaaS).** El modelo de "plataforma como servicio" (*Platform as a service* o *PaaS*) se puede definir como un conjunto de plataformas compuestas por uno o varios servidores de aplicaciones y una base de datos (no todas las plataformas incluyen la posibilidad de tener la BBDD) que ofrecen la posibilidad de ejecución de aplicaciones (escritas en *java, ruby, python*, etc.). Dicho de otra manera, este modelo consiste en paquetes de sistema operativo y aplicaciones específicas virtualizadas en servidores. Las ofertas de PaaS pueden dar servicio a todas las fases del ciclo de desarrollo y pruebas del software, o pueden estar especializadas en cualquier área en particular, tal como la administración del contenido. Además, el proveedor será el encargado de escalar los recursos en caso de que la aplicación lo requiera, del rendimiento óptimo de la plataforma, seguridad de acceso, etc.

  Un ejemplo de PaaS es Google App Engine, que permite desarrollar y ejecutar aplicaciones Python propias en la plataforma de Google. Otro ejemplo de *PaaS* es 10gen, una plataforma pero que puede ser descargada como paquete Open Source para montar nuestro propio servicio *PaaS* sobre servidores Linux.

- **Infraestructura como Servicio (IaaS).** El modelo de infraestructura como servicio (Infraestructure as a service) ofrece almacenamiento básico y capacidades de cómputo como servicios estandarizados en la red. En él se incluyen todos los servicios de computación que utilizan virtualización para manejar tipos específicos de cargas de trabajo (máquinas virtuales específicas). Dichos servicios vienen acompañados por los de almacenamiento relacionado (bases de datos) y no relacionado (discos).

  El ejemplo más conocido en el mundo comercial es Amazon Web Services, cuyos modelos EC2 y S3 ofrecen cómputo y servicios de almacenamiento esenciales. De hecho, algunos de los estudios realizados para medir la eficacia del simulador en el que se apoya este proyecto, se basan en las anteriores soluciones de Amazon, como se verá más adelante.

La Figura 3 refleja los distintos modelos de prestación de servicios que ofrecen los proveedores cloud a los clientes. Estos modelos, como se describió antes, son dividen en tres grupos: SaaS, PaaS e IaaS.

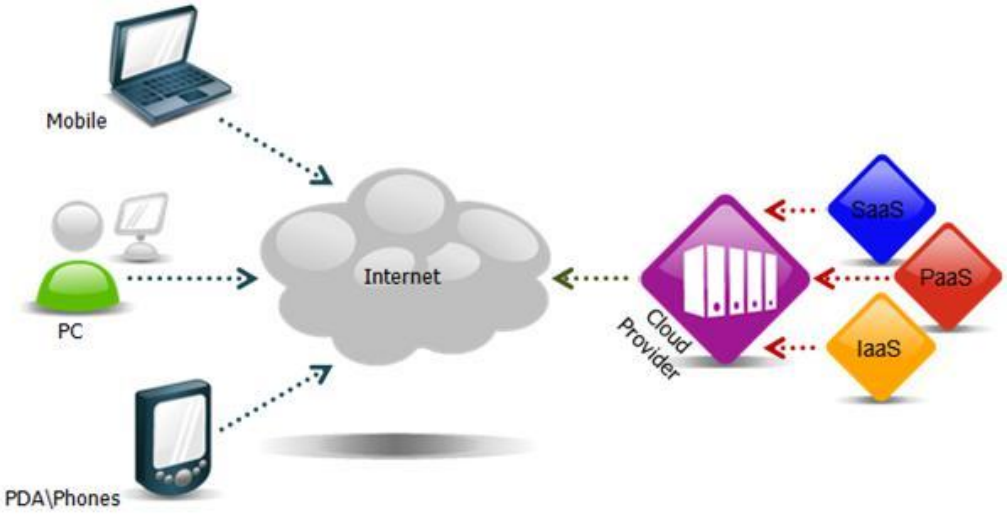

Figura 3: Integración de los modelos de cloud computing en la red

Los modelos representados en la Figura 3 se muestran de nuevo, aunque de manera detallada. En la Figura 4 aparecen reflejados los tres modelos de prestación de servicios, los distintos elementos que componen cada uno de ellos y las distintas relaciones que hay entre estos componentes.

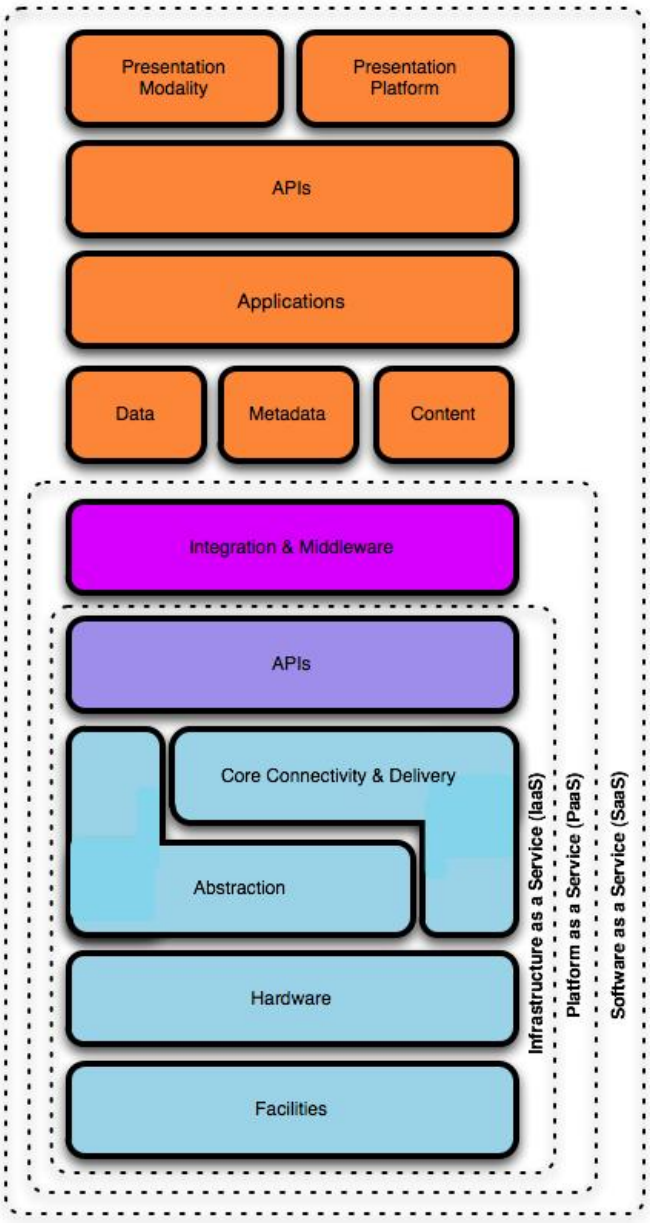

Figura 4: Modelos de prestación de servicios

Ninguno de estos modelos existe en un entorno aislado, por lo que es importante entender la estrecha relación entre ellos. IaaS es la base de todos los servicios en la cloud, con PaaS articulado encima de IaaS y SaaS, a su vez, articulado sobre IaaS. La Figura 5 ilustra la taxonomía de las soluciones disponibles en el Cloud Computing, según los distintos modelos explicados anteriormente.

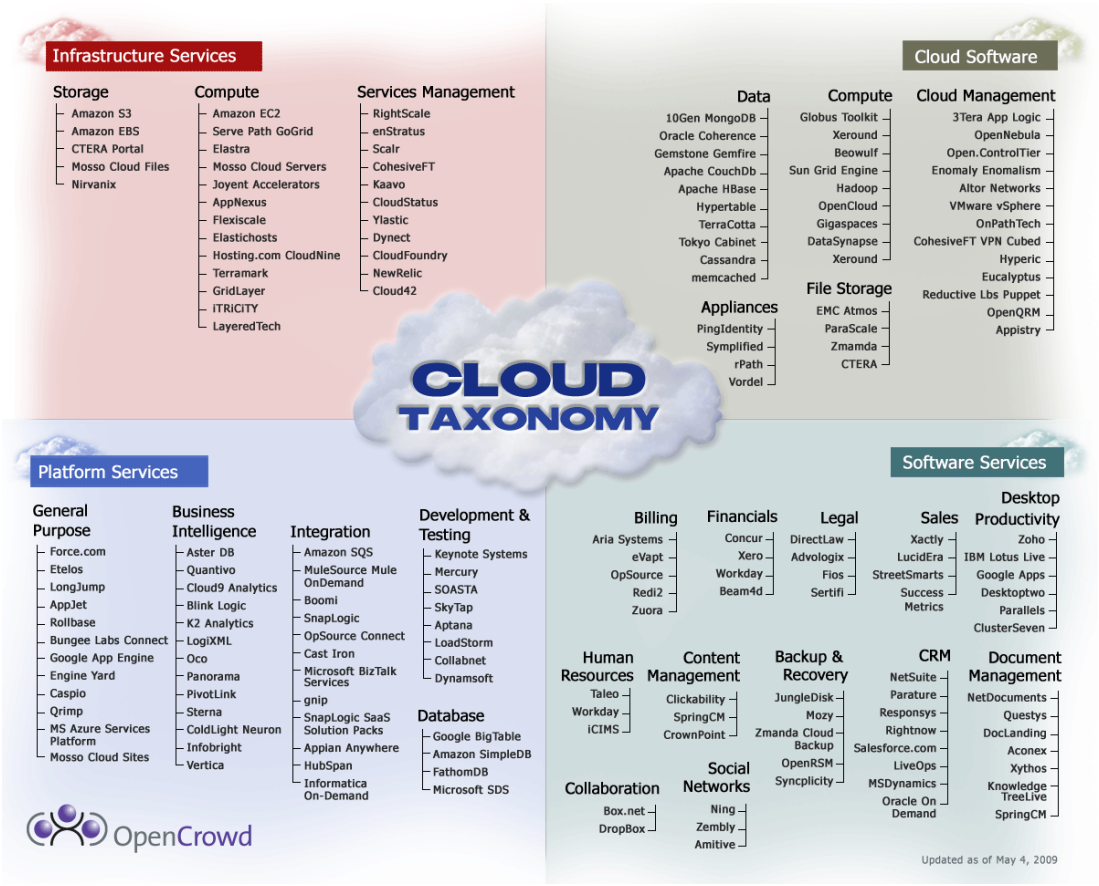

Figura 5: Taxonomía de servicios de Cloud Computing

Como puede observarse, el gráfico está dividido en las distintas capas que conforman los servicios en la nube y además, para cada uno de esos servicios, se ofrece una explicación de sus objetivos.

Por otro lado, volviendo al concepto de costes que se comentó anteriormente y que era tan significativo en este modelo de computación, sería beneficioso conocer qué modalidad es más común para cada modelo de prestación de servicios.

En el modelo "saas" se utiliza el pago por el potencial uso, donde la modalidad que más se utiliza es el pago por usuario y a veces se mezcla este con el pago por algunas funcionalidades adicionales, es decir, pago por usuario básico, avanzado y funcionalidad completa. Un ejemplo de esta modalidad es Zoho Creator.

En el modelo "paas" se utilizan los dos tipos de modalidades: pagos por el potencial uso, como por ejemplo Velneo (19€ /usuario) y por uso con Bungeeconnect (0.06 $/usuario por hora y por sesión). Por su parte, en el modelo "iaas", lo normal es utilizar la modalidad de pago por uso del recurso, ya sea máquina o CPU, GB de discos, tráfico de entrada y salida, etc.

De estos datos se deduce que la modalidad de pago por uso o consumo es la más utilizada cuando el modelo de prestación ofrecido opera a bajo nivel (sobre todo el iaas) mientras que los servicios de más alto nivel utilizan otras modalidades. En este proyecto, como se verá más adelante, se gestionarán todos los niveles del cloud computing, empezando desde los recursos de infraestructura, por lo que la modalidad de pago por uso o consumo se convierte en la opción principal a tener en cuenta.

2.1.4 Modalidades de despliegue y consumo

Independientemente de los modelos de prestación anteriormente descritos, se pueden distinguir cuarto modalidades de despliegue y consumo de los de servicios de computación en nube:

- **Nubes Privadas.** Las nubes privadas son propiedad de una organización o su correspondiente proveedor de servicios. Ofrecen un entorno operacional dedicado (single-tenant) con todos los beneficios, funcionalidad, escalabilidad y elasticidad propios del Cloud Computing.

  La infraestructura es propiedad y/o está físicamente localizada en una organización o su correspondiente proveedor de servicios, quien se encarga de la gestión de la nube y los controles de seguridad.

  Los consumidores de los servicios de las nubes privadas se consideran "de confianza" ("trusted") y son normalmente parte de la estructura legal y contractual de la organización (empleados, proveedores, socios comerciales, etc.). Los consumidores que no se consideran fiables son todos aquellos que pueden acceder a todos o algunos de los servicios pero que no forman parte de la organización.

- **Nubes Públicas.** Este tipo de nubes son propiedad de los proveedores de servicios externos y pueden ofrecer un entorno operativo tanto dedicado (single-tenant) como compartido (multi-tenant) con todos los beneficios, funcionalidad, escalabilidad y elasticidad propios del Cloud Computing.

  Las infraestructuras de las nubes públicas son también propiedad de dichos proveedores de servicios y se ubican físicamente en sus centros. Asimismo, la responsabilidad de su gestión y mantenimiento recae en estos proveedores.

  Todos los consumidores de las nubes públicas se consideran no fiables.

- **Nubes Gestionadas.** Este tipo de nubes son proporcionadas por los proveedores de servicios y pueden ofrecer un entorno operativo tanto dedicado (single-tenant) como compartido (multi-tenant) con todos los beneficios, funcionalidad, escalabilidad y elasticidad propios del Cloud Computing.

Las infraestructuras físicas son propiedad de la organización y se localizan físicamente en sus centros de datos. No obstante, la gestión y control de las mismas se deja en manos de los proveedores de servicios.

Los consumidores de dichas nubes pueden ser tanto fiables como no fiables.

- **Nubes Híbridas.** Las nubes híbridas combinan las características da las nubes públicas y las privadas, ofreciendo intercambio de información y posible portabilidad y compatibilidad de aplicaciones entre diferentes nubes, haciendo uso de metodologías estandarizadas o propietarias independientemente del propietario o localización.

  Los consumidores de dichas nubes pueden ser tanto fiables como no fiables.

2.1.5 Características principales del cloud computing

Una vez que se han enunciado los distintos modelos asociados al cloud computing, los tipos de nubes ofertadas por los proveedores y se ha comentado qué es la computación en nube y cómo surgió, se va a proceder a la explicación de las características principales del cloud computing, las cuales, aunque ya han sido enunciadas anteriormente, se recogen ahora de manera conjunta y organizada.

Como se indicó en la definción del cloud computing, la principal función de las nubes es usar "*[…] los recursos físicos disponibles para ofrecer al usuario servicios de computación bajo demanda, escalables y (a menudo) en un entorno multiusuario (multi-tenant) […]*" [1]. Esta definición abarca las características clave de cloud computing, que se describen con más detalle a continuación.

- **Las TI como servicio:** En primer lugar, todo lo que cloud computing es capaz de ofrecer, lo ofrece como servicio. Eso significa que los usuarios pueden consumir esos servicios a través de la nube sin necesidad de conocer el manejo de las complejas estructuras subyacentes o los recursos físicos que usan. Esta abstracción ofrece a las empresas la oportunidad de disminuir costes en equipos hardware y personal técnico encargado de su mantenimiento.

- **Escabilidad bajo demanda:** En segundo lugar, esos servicios se ofrecen bajo demanda, siguiendo el modelo de pago por uso. Los recursos se asignan dinámicamente para adaptarse a las necesidades de los usuarios en cada momento. Desde el punto de vista del consumidor, dichos recursos parecen ilimitados y pueden ser adquiridos en cualquier cantidad en todo momento.

- **Independencia del dispositivo y localización:** Además, los servicios son accesibles a través de Internet, por lo que los usuarios pueden acceder desde distintas localizaciones y dispositivos (PCs, PDAs, etc.).

- **Pago por uso:** La computación se ofrece como un servicio medible y facturable, de forma similar a la electricidad, el agua, o la telefonía.

- **Entorno multi-usuario:** Por último, permite compartir recursos y costes entre gran número de usuarios. Este hecho deriva en la mejora del proceso de negocio ya que las empresas comparten datos y aplicaciones, centrándose en el proceso de negocio en lugar de las infraestructuras y tecnologías que lo soportan. Al mismo tiempo, los usuarios tienen en todo momento a su disposición las últimas versiones de los sistemas y servicios que usan. El modelo multi-tenant (una única instancia compartida por múltiples usuarios) hace que a los proveedores les resulte mucho más fácil ofrecer acceso instantáneo a las nuevas características a todos los usuarios, en comparación con los modelos tradicionales.

2.1.6 Ventajas y desventajas del cloud computing

En este punto se va a pasar a describir cuáles son los beneficios e inconvenientes fundamentales de utilizar una infraestructura basada en cloud. Además de las características comentadas en el punto anterior se describen a continuación otros conceptos que pueden ser entendidos como ventajas dentro de la infraestructura del cloud computing:

- **Integración.** La tecnología de cloud computing se puede integrar con mucha mayor facilidad y rapidez con el resto de las aplicaciones de la empresa (tanto software tradicional como cloud computing basado en infraestructuras), ya sean desarrolladas de manera interna o externa.

- **Adaptabilidad.** Las infraestructuras de cloud computing proporcionan mayor capacidad de adaptación, recuperación de desastres completa y reducción al mínimo de los tiempos de inactividad.

- **No requiere hardware adicional.** Una infraestructura 100% de cloud computing no necesita instalar ningún tipo de hardware. Una ventaja importante de la tecnología de cloud computing es su simplicidad y el hecho de que requiera mucha menor inversión para empezar a trabajar.

- **Implementación más rápida y con menos riesgos.** Es posible empezar a trabajar muy rápidamente gracias a una infraestructura de cloud computing. No es necesario tener que esperar meses o años e invertir grandes cantidades de dinero antes de que un usuario inicie sesión en la nueva solución. Las aplicaciones en tecnología de cloud computing estarán disponibles en cuestión de semanas o meses, incluso con un nivel considerable de personalización o integración.

- **Actualizaciones automáticas**. Si actualizamos a la última versión de la aplicación, nos veremos obligados a dedicar tiempo y recursos (que no tenemos) a volver a crear

nuestras personalizaciones e integraciones. La tecnología de cloud computing no nos obliga a decidir entre actualizar y conservar nuestro trabajo, porque esas personalizaciones e integraciones se conservan automáticamente durante la actualización.

- **Uso eficiente de la energía.** En este caso, a la energía requerida para el funcionamiento de la infraestructura. En los datacenters tradicionales, los servidores consumen mucha más energía de la requerida realmente. En cambio, en las nubes, la energía consumida es sólo la necesaria, reduciendo notablemente el desperdicio.

Sin embargo, aunque este tipo de computación ofrece una buena cantidad de beneficios también existen una serie de inconvenientes a tener en cuenta, muchos de ellos derivados de la propia naturaleza del concepto. Algunos de estos inconvenientes son:

- **Dependencia del proveedor**. La centralización de las aplicaciones y el almacenamiento de los datos origina una dependencia de los proveedores de servicios.
- **Necesidad de acceso a la red**. La disponibilidad de las aplicaciones están atadas a la disponibilidad de acceso a internet.
- **Vulnerabilidad de la privacidad**. Los datos "sensibles" del negocio no residen, en muchas ocasiones, en las instalaciones de los clientes por lo que podría generar un contexto de alta vulnerabilidad para la sustracción o robo de información.
- **Seguridad en la red**. La información de la empresa debe recorrer diferentes nodos para llegar a su destino, cada uno de ellos (y sus canales) son un foco de inseguridad. Si se utilizan protocolos seguros, HTTPS por ejemplo, la velocidad total disminuye debido a la sobrecarga que requieren estos protocolos.
- **Escalabilidad a largo plazo**. A medida que más usuarios empiecen a compartir la infraestructura de la nube, la sobrecarga en los servidores de los proveedores aumentará, si la empresa no posee un esquema de crecimiento óptimo puede llevar a degradaciones en el servicio o jitter altos.

2.2 Simulación en Cloud Computing

Como hemos podido ver en el apartado anterior, el cloud computing ha emergido como la tecnología líder en la prestación de servicios de computación fiable, segura, tolerante a fallos, sostenible y escalable, la cual presenta sus servicios mediante los modelos anteriormente vistos: *SaaS, IaaS y PaaS*. Además, estos servicios pueden ser ofrecidos en los centros de datos privados (nubes privadas), pueden ser comercialmente ofrecidos por los clientes (las nubes públicas), y también es posible que las nubes públicas y privadas se combinen en las nubes híbridas.

Sin embargo, estas arquitecturas de nube tienen un enorme grado de complejidad. Además, surge una creciente demanda de tecnologías eficientes en cuanto a consumo de energía, de tiempo, y de metodologías que definan la evaluación de algoritmos, aplicaciones y políticas antes del proceso de desarrollo real. Debido a que la utilización de bancos de pruebas reales limita los experimentos a la escala del mismo y hace que la obtención de los resultados sea una tarea sumamente difícil, se requieren enfoques alternativos de experimentación que se reflejen en nuevas tecnologías de cloud.

Una alternativa adecuada es la utilización de herramientas de simulación, que permiten evaluar la hipótesis previa al desarrollo de software en un entorno donde se pueden reproducir las pruebas. Específicamente en el caso de computación en la nube, donde el acceso a la infraestructura se ve reflejada en pagos reales, la opción de utilizar soluciones basadas en simulación ofrece ventajas significativas, ya que permite a los clientes del propio cloud probar sus servicios en un entorno controlado de forma gratuita, y ajustar el rendimiento resultante antes de pasar a la solución real.

Desde el punto de vista del proveedor, los entornos de simulación permiten la evaluación de diferentes tipos de escenarios de consumo de recursos bajo carga y distribución de precios variables. Tales estudios podrían ayudar a los proveedores en la optimización de los costes de acceso a recursos, con especial atención sobre los beneficios de la mejora. En ausencia de plataformas de simulación, los clientes y los proveedores de Cloud tiene que confiar en evaluaciones teóricas e imprecisas, o en los casos de “prueba y error” que conducen a la prestación de un servicio ineficiente.

Así pues, el objetivo principal de este tipo de herramientas consiste en proporcionar un marco de simulación generalizada y extensible que permita la experimentación en las nuevas infraestructuras de cloud computing y sus servicios de aplicación. Mediante su utilización, investigadores y desarrolladores de la rama de actividad pueden centrarse en cuestiones específicas de diseño de los sistemas que quieren investigar, sin que se trate sobre los detalles de bajo nivel relacionados con infraestructuras y servicios del cloud.

A continuación, se van a describir algunas de las principales herramientas de simulación de cloud computing que existen en el mercado. Sin embargo, no se puede hablar de la simulación

de clouds sin mencionar previamente cuales fueron las antecesoras de este tipo de herramientas: *las herramientas de simulación de Grids*.

En la última década, los grids se han posicionado como la principal infraestructura para la prestación de servicios de alto rendimiento. Estos servicios van destinados a satisfacer importantes necesidades de cómputo o el manejo de gran cantidad de datos para fines científicos.

En apoyo a la investigación y el desarrollo de nuevos componentes, políticas y *middleware* para los grids, aparecieron en los últimos años varios simuladores de Grid, entre los que podríamos destacar los siguientes:

- **SimGrid** es un marco generalizado para la simulación de aplicaciones distribuidas en plataformas de Grid.
- **GangSim** es una herramienta de simulación de Grid que proporciona soporte para el modelado de organizaciones y recursos virtuales basados en Grid.
- Por su parte, **GridSim** es una herramienta de simulación (que funciona mediante eventos) que simula recursos de Grid heterogéneos. Es compatible con el modelado de entidades Grid, de usuarios, de máquinas y de las comunicaciones de red, incluyendo el tráfico de la misma.

A pesar de que todas estas herramientas anteriores son capaces de modelar y simular el comportamiento de las aplicaciones Grid (ejecución, programación, asignación y monitorización), en un entorno distribuido consistente en múltiples organizaciones de grids, ninguna de ellas es capaz de soportar la infraestructura y los requisitos a nivel de aplicación derivados de un paradigma de computación en nube.

En particular, existe muy poco o ningún soporte por parte de las herramientas de simulación Grid para modelar la gestión de recursos bajo demanda y la gestión de la aplicación. Además, el cloud se compromete a prestar servicios basados en suscripciones a un modelo de pago por consumo a los clientes del mismo. Por lo tanto, las herramientas de modelización y simulación de infraestructuras de Cloud deben proporcionar soporte para entidades económicas tales como agentes de bolsa o de intercambio que permitan el comercio en tiempo real de servicios entre clientes y proveedores. De entre los simuladores disponibles actualmente en el mercado que han sido mencionados anteriormente, sólo GridSim ofrece soporte para la gestión de recursos económicos y la simulación del programa de aplicación.

Otro aspecto relacionado con los clouds que se debe tener en cuenta es que la investigación y el desarrollo de sistemas, aplicaciones y servicios de cloud computing están en una etapa inicial. Hay un número importante de ideas que necesitan una investigación detallada en lo que se refiere a software de cloud. Entre los principales temas de interés de los desarrolladores de clouds se encuentran estrategias económicas para la prestación de recursos virtualizados a las solicitudes de los clientes, la programación de las aplicaciones, el

descubrimiento de recursos, las negociaciones entre clouds, la creación de organizaciones de clouds, etc.

Para solucionar toda esta problemática surge, entre otros, un simulador que evoluciona a partir de Gridsim y que se conoce como *CloudSim*:

- La herramienta de simulación **CloudSim** tiene como objetivo principal proporcionar un marco de simulación generalizada y extensible que permita el modelado, la simulación y prueba de las nuevas infraestructuras y servicios de Cloud Computing, permitiendo a los investigadores y usuarios de la mismas centrarse en aspectos específicos de diseño del sistema que quieren investigar, sin que tengan que preocuparse por detalles de bajo nivel relacionados con la gestión de las infraestructuras y los servicios propios del cloud.

Por último, en este apartado se realizará una introducción a la herramienta **SIMULACRUM**. Esta herramienta es utilizada en la simulación de entornos de experimentación en cloud computing y tiene como objetivo principal gestionar las características de la topología de la red, las cuales tienen un papel vital en este tipo de simulaciones.

2.2.1 SimGrid

SimGrid es un conjunto de herramientas que proporciona funcionalidades básicas para la simulación de aplicaciones distribuidas en entornos a gran escala y de carácter heterogéneo.

El desarrollo eficiente de aplicaciones concurrentes en este tipo de entornos plantea numerosos desafíos:

- Entender el comportamiento de la ejecución del código no es un asunto trivial.
- La realización de experimentos en plataformas en el mundo real a gran escala tampoco es trivial. Esto se debe principalmente a tres razones fundamentales: por un lado se requiere una implementación totalmente funcional, el proceso de experimentación está limitado a unas cuantas configuraciones de plataformas y éste en muchos casos resulta no repetible.
- Es muy difícil conseguir unos resultados de las simulaciones precisos. Esto se debe a que un mayor grado de precisión en la simulación implica un consumo de tiempo mucho mayor.
- El código simulado es código poco trabajado y puede diferir del código real del software.

La herramienta SimGrid aborda todos los problemas mencionados a través de una

infraestructura software “multi-componente” para el prototipado, desarrollo y despliegue de las aplicaciones.

De esta forma, las características principales que definen a la herramienta de simulación SimGrid son:

- Rapidez y capacidad de simulación de gran precisión (SURF).
- Existe la posibilidad de ejecutar el mismo código en modo de simulación total o parcial o en modo natural (GRAS, SMPI).
- Contiene una API para el prototipado de aplicaciones de manera rápida con el fin de probar y evaluar algoritmos distribuidos (MSG), aunque esta característica solo está disponible en modo de simulación.
- Contiene otra API para el desarrollo de aplicaciones orientadas a obtener código de aplicación rápido, robusto y portable. Esta característica está disponible tanto en modo de simulación como en modo normal.
- Una API para simulación de aplicaciones MPI (Message Passing Interface) para estudiar los efectos de una plataforma heterogénea (SMPI). Esta característica está disponible únicamente en el modo de simulación parcial.

SimGrid ofrece varios entornos de programación construidos sobre un núcleo de simulación único. Cada entorno se dirige a un cliente específico y constituye un paradigma diferente. Para elegir cuál de ellos se desea utilizar, es necesario tener en cuenta lo que se desea hacer y cuál sería el resultado de dicho trabajo.

Si queremos estudiar un problema teórico y comparar varias heurísticas, lo normal es utilizar MSG (Generic Application Simulation). Fue diseñado para conseguir el prototipado de aplicaciones y algoritmos distribuidos de forma rápida. Por ello, la simulación de situaciones reales no es el objetivo principal de este entorno y los aspectos técnicos más molestos de plataformas reales están abstraídos en este modelo.

Si lo que queremos es estudiar el comportamiento de una aplicación MPI utilizando técnicas de simulación, el modelo elegido debe ser SMPI (MPI Application Simulation).

Por otro lado, si lo que se desea desarrollar es una aplicación real distribuida, la mejor opción es elegir el modelo GRAS (Distributed Application Simulation and Deployment). Este tiene una completa API para la realización de aplicaciones distribuidas.

Todos estos modos de simulación se ven reflejados en la arquitectura de SimGrid, la cual aparece reflejada en la Figura 6:

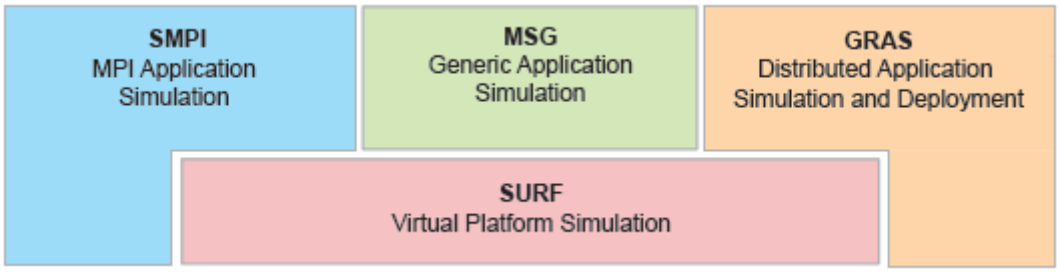

Figura 6: Arquitectura de SimGrid

Como se puede observar en esta arquitectura, los tres entornos de simulación (SMPI, MSG y GRAS) se apoyan sobre el SURF que constituye el eje de la aplicación.

De las características propias de cada una de estas interfaces se puede deducir que existen numerosas diferencias entre ellas. En lo que se refiere a concurrencia se pueden distinguir los siguientes aspectos:

- En MSG, todos los procesos de simulación de la aplicación ejecutan en un mismo proceso.
- Por su parte en GRAS, subconjuntos de procesos de la aplicación simulada ejecutan en múltiples procesos alojados en múltiples hosts, con el fin de incrementar la escalabilidad.
- Finalmente en SMPI, cada proceso de aplicación ejecuta como un proceso independiente.

Algunos ejemplos de escenarios en los que es común encontrar la utilización de esta herramienta son los siguientes:

- Cuando nos encontramos ante un sistema paralelo y lineal que funciona sobre un clúster básico.
- En cualquier aplicación paralela que está ejecutando en una red de estaciones de trabajo.
- En un proceso de simulación científica que está ejecutando en una plataforma grid de gran complejidad.
- En cualquier aplicación de monitorización de redes que está ejecutando en una red amplia.
- Cuando contamos con una aplicación de compartición P2P corriendo en servidores de Internet volátiles.

A continuación, se va a describir otra herramienta de simulación de grids como GridSim, la cual mejora las prestaciones ofrecidas por SimGrid.

2.2.2 GridSim

Como acabamos de ver, la herramienta de simulación SimGrid, es una herramienta basada en lenguaje C para la simulación aplicaciones programadas. Soporta el modelado de recursos de tiempo compartido y donde la carga puede ser introducida como constantes o de trazas reales.

Es una herramienta muy potente que permite la creación de tareas en relación a su tiempo de ejecución y de recursos con respecto a la capacidad de una máquina estándar. A través del API de SimGrid, las tareas pueden ser asignadas a los recursos de acuerdo a la política que esté siendo simulada. Además, ha sido utilizada por un importante número de estudios reales y ha sido demostrada de sobra su capacidad de simulación.

Sin embargo, el hecho de que SimGrid esté restringida a una única entidad de programación y a sistemas de tiempo compartido, hace difícil simular múltiples usuarios, aplicaciones y programaciones (con sus políticas propias), sin extender la herramienta. Además, muchos recursos en el entorno Grid son máquinas de espacio compartido y que necesitan apoyo en las simulaciones. La herramienta de simulación GridSim recoge las ideas de los sistemas existentes y las mejora, superando así las limitaciones encontradas.

La herramienta de simulación *GridSim* proporciona una gran facilidad para la simulación de diferentes clases de entidades distintas, como pueden ser recursos, usuarios, aplicaciones, brokers de recursos y programas. Se puede utilizar, entre otras cosas, para simular la programación de aplicaciones de sistemas de computación distribuidos de dominio administrativo, como clusters o grids.

Los programadores de aplicaciones en entornos Grid, llamados también “resource brokers”, llevan a cabo el descubrimiento de los recursos, así como la selección y la agregación de un conjunto variado de recursos distribuidos para el usuario. Esto significa que cada usuario tiene su propio resource broker privado y por ello puede estar dirigido a optimizar los requerimientos y objetivos de su propietario. Por el contrario, los programadores, gestionando recursos como clusters en un dominio administrativo simple, tienen control completo sobre la política utilizada en la asignación de recursos.

Esto significa que todos los usuarios necesitan presentar su trabajo a la central de planificación, la cual puede ser objeto de acciones de optimización global, tales como la utilización de sistemas superiores y la satisfacción global del usuario dependiendo de la política de asignación de recursos u la optimización para usuarios prioritarios.

Así, las características principales que definen a la herramienta GridSim son las que se recogen a continuación:

- Permite el modelado de tipos de recursos de distinta naturaleza (usuarios, aplicaciones, etc.)
- Los recursos pueden ser modelados bajo un marco de espacio y tiempo compartido.
- La capacidad de recursos puede ser definida (como MIPS o como referencia SPEC).
- Los recursos pueden estar localizados en cualquier espacio de tiempo.
- Los periodos festivos y vacaciones pueden ser mapeados de acuerdo

2.2.3 CloudSim

CloudSim se desarrolla en el Laboratorio de Cloud Computing y Sistemas Distribuidos (CLOUDS), del Departamento de Ingeniería del Software y Ciencías de la Computación de la Universidad de Melbourne. Se trata de una herramienta que tiene como objetivo principal proporcionar un marco de simulación generalizada y extensible que permita el modelado, la simulación y prueba de las nuevas infraestructuras y servicios de Cloud Computing.

Esto permite que los usuarios de la herramienta (principalmente investigadores) se centren en aspectos específicos de diseño del sistema que quieren investigar, sin que tengan que preocuparse por detalles de bajo nivel relacionados con la gestión de las infraestructuras y los servicios propios del cloud.

La Figura 7 muestra la arquitectura característica de la herramienta de simulación CloudSim y sus principales componentes:

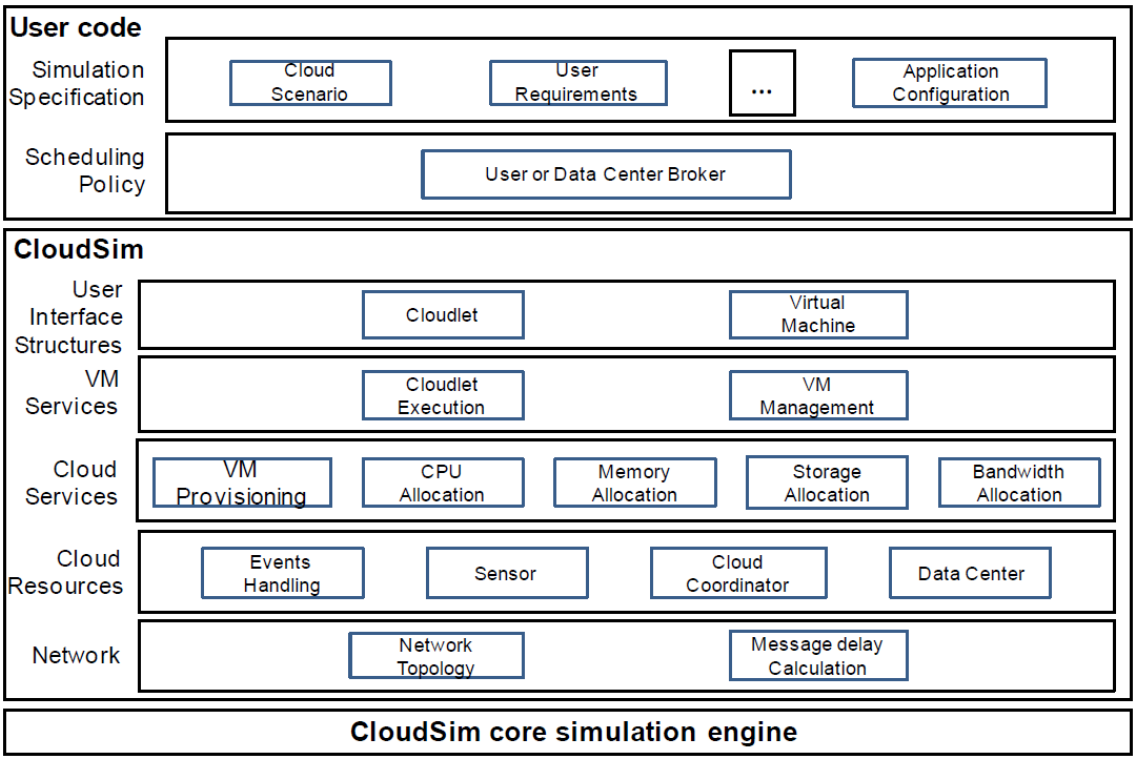

Figura 7: Arquitectura de CloudSim

Como puede observarse en el esquema anterior, en la capa más baja se encuentra el motor de simulación de eventos (o SimJava) que implementa el núcleo de funcionalidades requeridas por frameworks de simulación tales como listas y procesamiento de eventos, creación de los componentes del sistema, comunicación entre componentes y la gestión del reloj del sistema.

A continuación se encuentran librerías de la herramienta GridSim, que dan soporte a componentes software de alto nivel para el modelado de múltiples infraestructuras de Grid, incluyendo redes y perfiles de tráfico asociados, y principalmente componentes del Grid como recursos, conjuntos de datos, cargas de trabajo y servicios de información.

CloudSim se implementa en la siguiente capa, ampliando las funcionalidades básicas proporcionadas por la capa de GridSim. CloudSim proporciona un nuevo soporte para el modelado y simulación de entornos virtualizados basados en clouds, tales como interfaces de administración de máquinas virtuales, memoria, almacenamiento y ancho de banda.

La capa CloudSim gestiona la creación de instancias y la ejecución de las entidades principales (máquinas virtuales, *hosts, data centers*, aplicaciones) durante el transcurso de la simulación. Esta capa es capaz de instanciar de forma simultánea y gestionar de modo transparente cualquier infraestructura cloud compuesta por incluso cientos de componentes. Cuestiones fundamentales como la gestión de la ejecución de las aplicaciones y la monitorización

dinámica son manejadas por esta capa. Un proveedor de cloud que quisiera estudiar la eficacia de diferentes políticas de asignación, necesitaría implementar sus estrategias en esta capa, extendiendo la funcionalidad del núcleo.

Por último, la capa superior de esta pila corresponde con el código de usuario que comprende funcionalidades de configuración para hosts (número de máquinas, su especificación,…), aplicaciones (número de tareas y sus requisitos), máquinas virtuales, número de usuarios y sus tipos de aplicación, etc. Un desarrollador de aplicaciones en clouds puede crear un conjunto de requisitos de usuario, configuraciones y escenarios disponibles en esta capa, y de esta forma realizar pruebas sólidas basadas en configuraciones de clouds personalizadas soportadas dentro de CloudSim.

Si la computación en cloud se sitúa como un área de investigación que evoluciona rápidamente, existe una falta de estándares, herramientas y métodos ya definidos, que puedan abordar de manera eficiente problemas complejos a nivel de infraestructura y al nivel de aplicación. Por ello se estima que en los próximos años haya un importante esfuerzo de investigación, tanto en el ámbito académico como empresarial, para definir nuevos algoritmos de base, políticas y la evaluación de aplicaciones basada en contextos.

Al extender las funcionalidades básicas expuestas en CloudSim, los investigadores podrían realizar pruebas basadas en escenarios y configuraciones específicas, permitiendo así una mejor práctica en los aspectos más críticos del cloud computing.

Los principales aspectos que caracterizan a este importante simulador sobre otros son los siguientes:

- Soporta el modelado y la simulación a gran escala de data centers.
- Soporta el modelado y la simulación de servidores hosts virtualizados, con políticas personalizables para la provisión de recursos de host a las máquinas virtuales.
- Soporta el modelado y la simulación de recursos computacionales de energía.
- Soporta el modelado y la simulación de clouds federados.
- Soporta la inserción dinámica de elementos de la simulación, así como la detención y reanudación de la misma.
- Soporta políticas definidas por el usuario para la asignación de hosts a las máquinas virtuales y políticas para la asignación de los recursos de hosts a las máquinas virtuales.

2.2.4 SIMULACRUM

SIMULACRUM (SImulation pLAtform CReation and User-guided Modification) es una herramienta utilizada en la simulación de entornos de experimentación en cloud computing que tiene como objetivo principal gestionar las características de la topología de la red, las cuales tienen un papel vital en este tipo de simulaciones.

Esta herramienta se basa en un proceso de generación modular que puede ser adaptado a la salida deseada. Utiliza una serie de mecanismos (filtros) para ir desde un modelo de red a un entorno completamente definido que cumpla las necesidades del usuario.

El proceso de generación del modelo final se realiza siguiendo una serie de pautas, las cuales aparecen reflejadas en el siguiente flujo de datos:

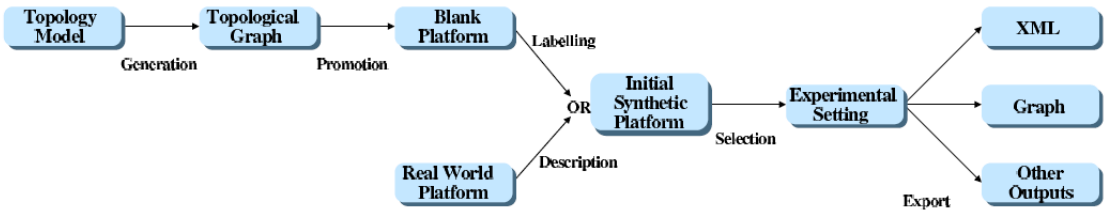

Figura 8: Flujo de generación de la herramienta SIMULACRUM

Como puede apreciarse en la Figura 8, el flujo de generación de la salida deseada consta de los siguientes pasos principales:

1. **Generación del grafo topológico**: Para generar un grafo topológico, SIMULACRUM se basa en varios modelos. La herramienta dispone de topologías clásicas como las de anillo o estrella. Pero además, la herramienta implementa modelos que propagan los nodos sobre un área cuadrada y conecta dos nodos cualesquiera *u* y *v* con una probabilidad *P(u,v).*

2. **Conversión de nodos (Promoción)**: En este paso se convierten los nodos abstractos de nuestro grafo en recursos de computación (hosts y clusters) y de red (routers).

3. **Etiquetado de ejes**: Una vez que el grafo topológico ha sido convertido a un conjunto final de recursos, determinadas propiedades de la comunicación (como latencias y anchos de banda) han de ser asociadas a los ejes de dicho grafo. Este proceso sigue una serie de reglas como pasaba en el caso de los nodos.

4. **Selección del subconjunto del grafo**: SIMULACRUM proporciona dos maneras de seleccionar un subconjunto de la plataforma inicial: o bien generándolo directamente o bien como una abstracción de un grid existente. La primera opción permite al usuario generar un entorno totalmente aleatorio, mientras que la segunda opción permite seleccionar un modelo de computación grid de los que están ya desplegados.

5. **Exportación**: El último paso del proceso de generación permite al usuario realizar algunos ajustes finales. La última versión de SIMULACRUM exporta una representación en formato XML del entorno experimental generado o muestra gráficamente la plataforma utilizando otras herramientas.

Con esta herramienta de gestión de las topologías de red finaliza el apartado del proyecto dedicado a las herramientas relacionadas con la simulación de clouds. A continuación, se realizará un análisis paso a paso de las necesidades del sistema a desarrollar.

Capítulo

3

3. Análisis del sistema

En este capítulo se va a realizar un análisis del sistema que se pretende desarrollar. Para ello, y tras una breve introducción, se comenzará con una descripción de la aplicación, de la cual se obtendrán una serie de requisitos de usuario que deberá cumplir el sistema y un modelo inicial a partir del cual se podrá trabajar en el posterior diseño del mismo. Finalmente se describirá detalladamente cada una de las funcionalidades encontradas y se representarán gráficamente mediante la utilización de los denominados "casos de uso" del sistema.

3.1 Introducción

El sistema que se pretende desarrollar, conforme a la información descrita en apartados anteriores, consiste en una aplicación cuyo cometido principal es el de interactuar con el simulador de cloud computing llamado "iCanCloud", tanto para proporcionarle datos de configuración como para interpretar los resultados obtenidos como producto de la simulación.

En un escenario habitual de configuración del cloud, un cliente no conoce inicialmente los requisitos a nivel de recursos que necesita para dar infraestructura al software alojado, y mucho menos, de forma óptima. En estos casos, la solución al problema consiste en lanzar una batería de pruebas variando los parámetros requeridos. Este hecho, al fin y al cabo, consiste en contratar recursos de la nube para pruebas, lo que conlleva irremediablemente un gasto para el cliente. Con el uso de la herramienta, este gasto se ve reducido de forma significativa, de forma que el cliente pueda realizar diversas simulaciones de su futura infraestructura hasta llegar al resultado que le resulte más provechoso en relación coste / tiempo.

La herramienta a desarrollar **tiene como objetivo hacer de este proceso de simulación una labor sencilla y de fácil manejo para el usuario, además de proporcionarle información útil de cara a una posterior toma de decisiones**.

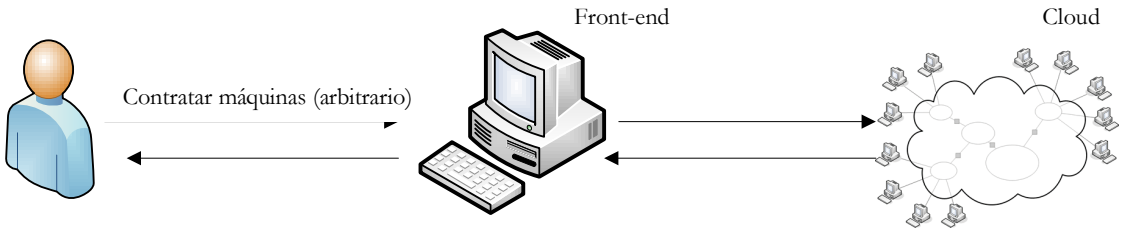

Figura 9: Escenario de prueba sin simulación

En el primer caso (pruebas en la nube sin proceso de simulación) representado en la Figura 9 anterior, observamos cómo un usuario debe acceder a un terminal que tenga acceso al proveedor de la nube y realizar en él una petición de una serie de máquinas, las cuales tendrán un determinado coste. Cada prueba, empleará una cantidad de tiempo determinada que el usuario no conoce a priori. El proceso de configuración por parte del usuario en este entorno consiste en realizar un número determinado de pruebas, interpretando los resultados en términos de tiempos y costes, y realizando nuevas peticiones variando los parámetros de entrada, hasta tener un número N suficientemente grande de resultados como para tomar decisiones sobre la configuración a emplear. Ahora bien, esto no implica que la configuración escogida por el usuario sea la óptima, sino la mejor con respecto a las pruebas realizadas.

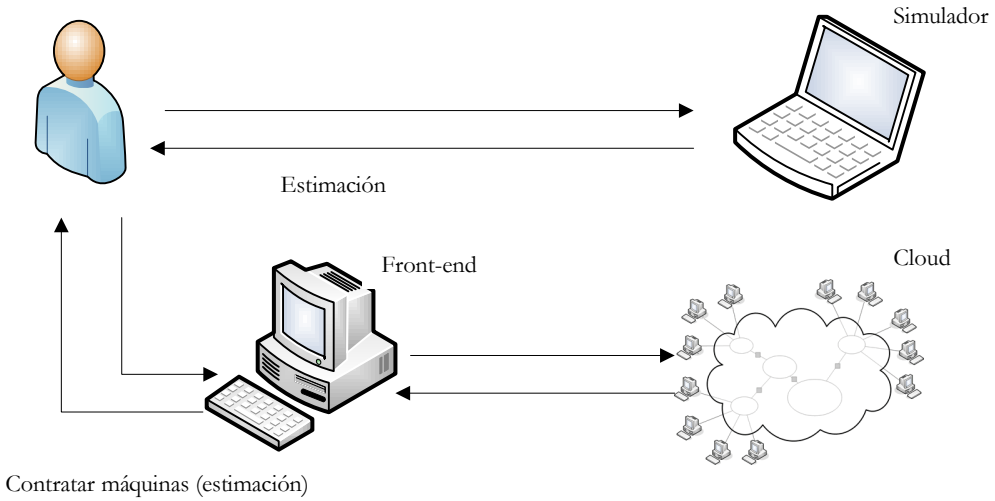

Figura 10: Escenario de prueba utilizando el simulador

En un escenario en el que el usuario utilice un proceso de simulación, como el que aparece reflejado en la Figura 10, la herramienta le devolverá una estimación de las máquinas que es necesario reservar en el cloud. De esta forma, el usuario podrá acceder al front-end anterior para solicitar el número de máquinas que le ha indicado el simulador, sin la necesidad de lanzar un número N de pruebas (como pasaba en el caso anterior). Así, el tiempo empleado en configurar el cloud es muchísimo menor al utilizado en el caso anterior. Pero no solo eso, ya que para el usuario el coste que acarrearán estas pruebas será significativamente menor (al no estar utilizando los servicios del cloud propiamente).

A continuación se va a pasar a describir el escenario que deberá cubrir el futuro sistema, especificando cuáles son los componentes que forman el mismo para después obtener un modelo sobre el que se va a trabajar.

3.2 Descripción de la aplicación

Como se comentó en apartados anteriores, la aplicación a desarrollar consiste en un gestor de entornos de configuración para la práctica de simulaciones en escenarios de computación en nube (cloud computing). Así, uno de los objetivos principales a cumplir es facilitar la labor de generación y configuración de **clouds** (nuestros escenarios de computación). Este proceso de configuración del cloud se traduce en la gestión de todos los recursos que conformarán el cloud, tanto a nivel de hardware como de software. Estos recursos son principales de dos tipos: las **máquinas virtuales** que representan el componente hardware, y **las tareas** a ejecutar, las cuales conforman el componente software.

Una máquina virtual no es sino una entidad abstracta que emula el comportamiento de un equipo físico y que puede ejecutar aplicaciones como si de una máquina real se tratase. Ahora bien, la principal característica de las máquinas virtuales consiste en que las aplicaciones ejecutadas en ellas están limitadas por los recursos proporcionados por estas mismas máquinas. Por ello, toda máquina virtual tiene una serie de recursos asociados (recursos hardware), que en nuestro caso desempeñarán el papel de componentes de estas máquinas.

El simulador “iCanCloud”, con el fin de llevar a cabo el diseño de estos entornos de cloud, proporciona un tipo de estructuras físicas modeladas con las que el usuario puede trabajar. Estas estructuras tienen como objetivo servir de capa de abstracción a los recursos de computación ofrecidos por el propio simulador. Las estructuras de las que hablamos no son sino estas “máquinas virtuales”, que pueden ser configuradas totalmente por el usuario. Por lo tanto, cada máquina virtual contiene una serie de componentes que la definen (en lo que se refiere a capacidad de cómputo, cantidad de memoria asignada, etc.) y que las hacen distintas unas de otras.

Figura 11: Representación jerárquica de componentes del sistema

Como se comentó anteriormente, además de estas máquinas virtuales, existe otro componente que define el comportamiento de los clouds de nuestra aplicación: las **tareas o trabajos**. Una tarea es una descripción del conjunto de aplicaciones software que queremos ejecutar en nuestro cloud. Si bien antes la máquina virtual era la unidad mínima de modelado del componente hardware del cloud, en el caso del componente software, cada tarea puede estar formada por una o varias **aplicaciones** que se podrán ejecutar un número determinado de veces. Así, la gestión de estas tareas y de las aplicaciones que las forman es otro de los objetivos principales del presente proyecto.

En la Figura 11 se muestra un esquema de los distintos elementos que formarán el sistema resultante de mayor a menor nivel de abstracción. Todo usuario de la aplicación podrá comunicarse con el sistema de simulación a través de la herramienta a desarrollar, con lo que podrá realizar diversas acciones relacionadas con la gestión de:

- Simulaciones
- Clouds
- Máquinas virtuales y hardware asociado

- Tareas o trabajos
- Aplicaciones

3.3 Descripción del modelo

En el apartado anterior se describieron las características del futuro sistema y se definieron los elementos que formarán parte del mismo. A continuación se comentará cómo encajan todos estos componentes en la futura aplicación mediante la elaboración de un modelo de datos que simbolice el escenario real del problema.

Como se ha comentado a lo largo de este documento, el sistema a desarrollar tiene por objetivo principal la gestión de entornos de simulación. Al tratarse de escenarios experimentales, el usuario no trabajará con componentes reales, sino con abstracciones o representaciones de los mismos. Se hace necesario de esa manera realizar un modelo para cada uno de los elementos que forman parte del sistema.

De manera resumida, el modelo debe recoger lo siguiente: todo usuario debe poder realizar una serie de simulaciones, donde cada una de ellas se realizará sobre un cloud previamente definido por el usuario, el cual a su vez estará compuesto por un número determinado de máquinas virtuales que le den soporte y una serie de tareas a realizar.

La Figura 12 muestra el modelo de datos básico que ha sido elegido para representar el nuevo sistema. Como se puede observar en el esquema, cada usuario de la aplicación puede interactuar con la aplicación para realizar una serie de simulaciones. Toda simulación requiere unos datos de entrada. Estos parámetros (el escenario de partida) se abstraen dentro de la entidad que llamaremos ‘cloud’. A su vez, como resultado de la ejecución de cada escenario, obtendremos una serie de resultados, lo que hemos denominado ‘informe’. Estos informes, recopilarán información en materia de tiempos y costes obtenidos por el simulador como producto de la ejecución del escenario de entrada.

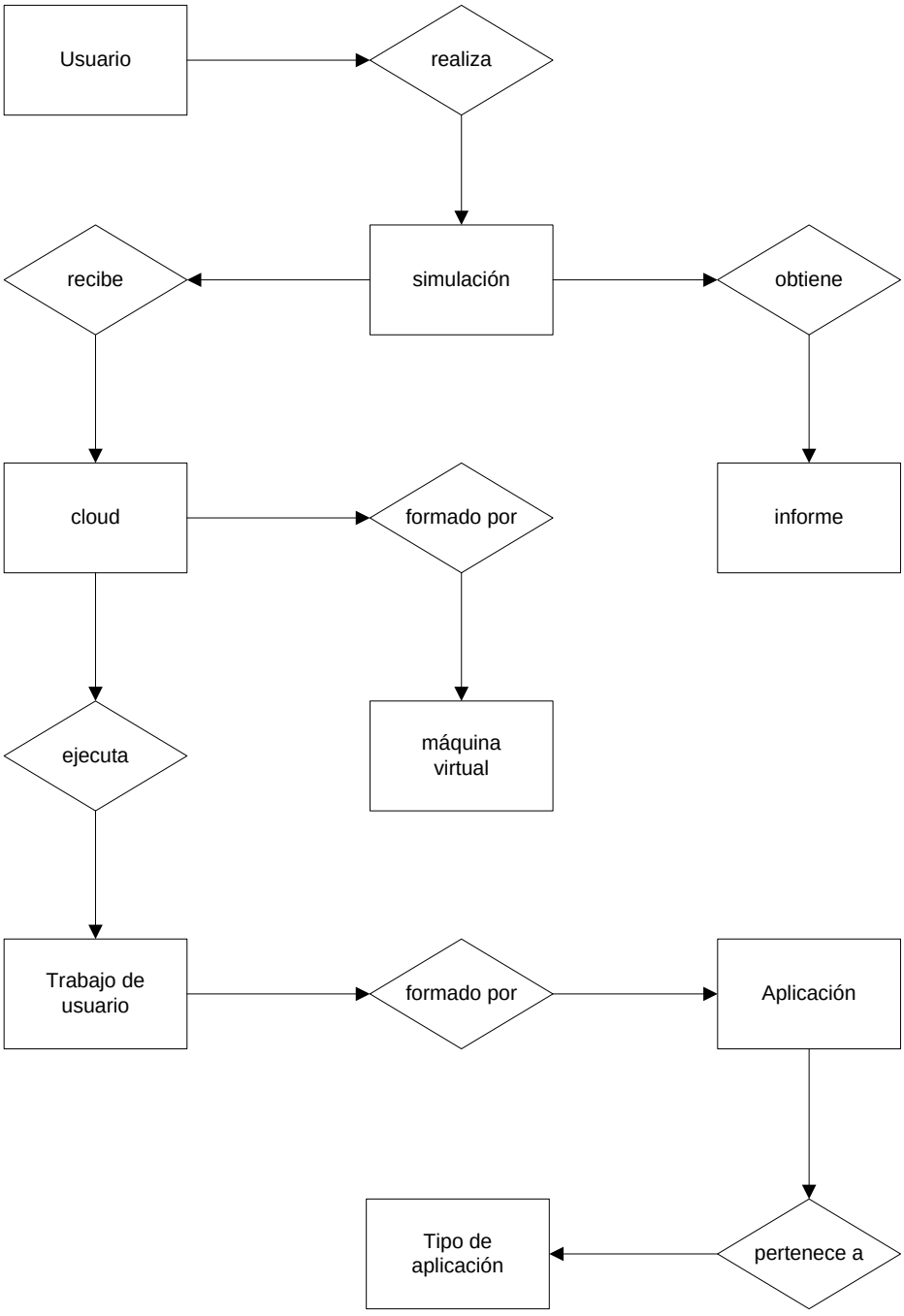

Figura 12: Modelo entidad-relación asociado al problema

De esta manera, el sistema deberá gestionar estos datos de entrada y salida de las simulaciones realizadas.

Por otro lado, cada uno de los escenarios de entrada (los clouds) estarán formados por una serie de máquinas virtuales previamente definidas (o modelos que definen máquinas virtuales reales) y por un conjunto de tareas o trabajos de usuario que deberán ser ejecutados por estas máquinas virtuales.

Cada modelo de máquina virtual (en adelante MV) debe representar fielmente una solución real. Existen multitud de MVs en el mercado, pero todas y cada una de ellas están limitadas por los recursos hardware asociados a las mismas. Así, el rendimiento de una MV depende de factores como el ancho de banda, la memoria asociada, etc. En el sistema se ha decidido identificar a cada modelo de máquina en base a cuatro componentes fundamentales:

- **CPU**: su capacidad de cálculo y el número de núcleos asociados.
- **RAM**: la capacidad de memoria volátil.
- **Almacenamiento**: la capacidad de memoria no volátil de cada disco duro así como el número de discos con los que cuente la máquina.
- **Hardware de entrada/salida**: tasas de transferencia de datos entre máquinas.

Además de las MVs, en el sistema se hace necesario representar las tareas a ejecutar por cada una de estas máquinas. Estas tareas se denominan también 'trabajos de usuario'.

El objetivo de este componente es permitir que el usuario defina el trabajo que quiere que se ejecute en el cloud. Al fin y al cabo, todo lo que ejecuta en el cloud son aplicaciones, de modo que cada trabajo estará compuesto por un conjunto de aplicaciones que ejecutarán sobre un conjunto de máquinas virtuales. Así, los componentes que forman cada trabajo de usuario son los siguientes:

- **Número de máquinas virtuales**: el número de máquinas virtuales, de entre las ofrecidas por el proveedor del cloud, que se necesitan para ejecutar el trabajo.
- **Identificador de máquina virtual**: marca el modelo de máquina virtual que se va a utilizar para la ejecución de las aplicaciones.
- **Identificador de aplicación**: la aplicación que va a correr sobre las máquinas virtuales seleccionadas.
- **Número de iteraciones**: determina en cuántas ocasiones se va a lanzar la aplicación escogida sobre las máquinas virtuales.

Por lo tanto, se puede entender una tarea o trabajo como una representación del conjunto de aplicaciones que un usuario desea ejecutar sobre un determinado cloud (imaginemos, por

ejemplo, que en el sistema están definidas las aplicaciones A, B, C y D, pero el usuario "Administrador" solo quiere lanzar una prueba con las 3 primeras).

Como se ha comentado antes, las tareas están formadas por un conjunto de aplicaciones a ejecutar. Éstas tienen a su vez una serie de parámetros de entrada, lo que permite que se configure su comportamiento de modo flexible.

Ahora bien, las aplicaciones software que pueden ejecutar estas máquinas virtuales dentro de un cloud pueden ser de naturaleza totalmente distinta. Existen millones de aplicaciones software en el mercado, distintas unas de otras, por lo que contar con un modelo de representación genérico resulta una tarea imposible.

Para que una aplicación software pueda ser simulada, es necesario crear un modelo matemático que la represente y trabajar a partir de dicho modelo. Así, cualquier software es susceptible de ser simulado en la herramienta siempre y cuando se cuente con un modelo que abstraiga su comportamiento.

Por este motivo, el simulador iCanCloud trabaja hasta la fecha con tres modelos de aplicaciones básicas distintas, donde cada una es totalmente parametrizable. Estos tipos de aplicaciones son:

- **DummyApplication**: se trata de una aplicación vacía, la cual se utiliza cuando no se ejecuta ninguna otra en el simulador.
- **LocalApplication**: se trata de una aplicación que se ejecuta en el nodo local. Principalmente lo que hace es leer datos de disco, tratarlos y volver a escribirlos en disco. Todos los parámetros son totalmente configurables por parte del usuario. Así, podríamos crear una aplicación de este tipo que leyera, por ejemplo, 10 MB, computara 100000 instrucciones y escribiera 500 KB, y esto un número "n" determinado de iteraciones.
- **BasicApplication**: se trata de una aplicación idéntica a una LocalApplication con la salvedad de que sus parámetros no son configurables, sino que utiliza unos valores introducidos por defecto.

En principio, es posible modelar cualquier tipo de aplicación en el sistema, siempre y cuando se genere un modelo matemático que lo represente (similar a las anteriores). Por ello, la aplicación debe ser capaz de leer de forma dinámica la información referente a los modelos de aplicaciones creados, de modo que aumentar el conjunto de aplicaciones con los que trabaja el sistema resulte una tarea trivial.

Todas las entidades descritas hasta el momento representan los elementos que forman un determinado cloud. Además de esto, ya sabemos que todo cloud está formado por un

componente hardware y otro software. Así, la trazabilidad de los elementos anteriores con el cloud se describe a continuación. En el sistema un modelo de cloud está formado por:

- **Infraestructura proporcionada por el proveedor**: se compone de un determinado número de máquinas virtuales ofertadas por el proveedor del cloud. Sobre estas máquinas se ejecutarán los trabajos que se requieran.
- **Tareas de usuario**: conjunto de trabajos definidos por uno o varios usuarios, que contienen a su vez el número y tipo de máquinas virtuales necesarias, la/s aplicación/es a ejecutar y el número de iteraciones.

Con estos conceptos como elementos principales del modelo que representa el sistema, se describirá a continuación cuál será la funcionalidad a desarrollar por el mismo, especificando los casos de uso derivados de la misma.

3.4 Funcionalidad a desarrollar

El modelo de datos de la Figura 12 introducía los principales conceptos que gestionará el sistema y deja al descubierto la necesidad de persistencia de dicha información. Por ello, se hace necesaria la gestión de un repositorio de elementos, que permitan al usuario mantener los datos con la que se ha trabajado en sesiones anteriores.

Los elementos más susceptibles de ser almacenados son aquellos que se utilizan como entrada de las simulaciones de la aplicación, por lo que el sistema deberá ofrecer la funcionalidad de almacenamiento y recuperación de los siguientes módulos:

- Repositorio de clouds
- Repositorio de máquinas virtuales
- Repositorio de tareas
- Repositorio de aplicaciones

Por su parte, los elementos que constituyen la salida de las simulaciones (los resultados obtenidos), también deberán poder ser almacenados para su posterior estudio. Por ello, el sistema deberá ofrecer la posibilidad de generar informes detallados de cada ejecución para su estudio y/o almacenamiento en disco.

Independientemente de estas funcionalidades de almacenamiento y recuperación de datos, existen un conjunto de funciones que deberán contemplarse en el sistema y que resumen el conjunto de acciones que todo usuario podrá realizar con la herramienta. Estas

funcionalidades aparecen clasificadas dependiendo del elemento al que vayan dirigidas y se dividen en siete grupos principales.

Las operaciones generales a implementar por el sistema, así como las operaciones básicas en las que se dividen las anteriores, aparecen reflejadas a continuación:

- Gestionar máquinas virtuales
    - Crear nueva máquina virtual
    - Borrar máquina virtual
    - Modificar máquina virtual
    - Borrar todas las máquinas virtuales
- Gestionar tareas
    - Crear nueva tarea
    - Borrar tarea
    - Modificar tarea
    - Borrar todas las tareas
- Gestionar aplicaciones
    - Crear nueva aplicación
    - Borrar aplicación
    - Modificar aplicación
    - Borrar todas las aplicaciones
- Gestionar clouds
    - Crear nuevo cloud
    - Borrar cloud
    - Modificar cloud
    - Borrar todos los clouds
- Gestionar simulaciones

    - Generar ficheros configuración
    - Lanzar simulación
    - Monitorizar simulación
- Gestionar resultados
    - Interpretar resultados
    - Generar gráficas
    - Generar informe
- Gestionar la configuración del usuario
    - Guardar configuración
    - Cargar configuración

Con este conjunto de servicios ofertados se cubre toda la funcionalidad descrita anteriormente y se cubren también todos los objetivos adelantados en el primer apartado de este documento. Así pues, pasaremos a detallar cada uno de estos servicios de manera adecuada en el siguiente punto.

3.5 Descripción de requisitos

En el anterior punto se enunciaban brevemente todas y cada una de las funcionalidades que va a incluir el sistema a desarrollar. A continuación se va a pasar a describir detalladamente en qué consiste cada una de ellas, comenzando por las operaciones generales para después pasar a describir aquellas operaciones más simples que forman parte de las anteriores.

Las principales funcionalidades a cubrir son las siguientes:

IDENTIFICADOR:	***RP-001***	TÍTULO:	Gestionar máquinas virtuales
DESCRIPCIÓN:	Como vimos anteriormente, el concepto de máquina virtual es el eje central de la configuración de nuestro entorno. Cada nube agrupa una serie de máquinas virtuales con un fin determinado. Por lo tanto, el sistema deberá incluir la opción de gestionar modelos correspondientes a estas máquinas virtuales y así asociarle una serie de recursos hardware que conformarán la infraestructura física de la nube.		

Tabla 1: Funcionalidad gestionar máquinas virtuales

IDENTIFICADOR:	***RP-002***	TÍTULO:	Gestionar tareas de usuario
DESCRIPCIÓN:	La infraestructura de cloud existe para ofrecer un conjunto de servicios de computación que realicen una serie de tareas. Estas tareas conforman el componente software de la nube. Por ello, el sistema debe dar la opción al usuario de gestionar modelos que representen estas tareas de acuerdo a una serie de parámetros como el número de iteraciones que ejecutará una determinada aplicación, o el número y tipo de máquinas en las que se ejecutará la misma.		

Tabla 2: Funcionalidad gestionar tareas

IDENTIFICADOR:	***RP-003***	TÍTULO:	Gestionar aplicaciones
DESCRIPCIÓN:	El sistema debe ofrecer la posibilidad de gestionar modelos que simulen el comportamiento de aplicaciones software de un determinado tipo. Es decir, a partir de un modelo matemático de un tipo de aplicación concreto (una app que calcule la distancia entre dos puntos, otra que calcule el cuadrado de un número), el sistema debe ser capaz de parametrizarlo con los valores que desee el usuario y lanzarlo sobre la infraestructura de cloud creada.		

Tabla 3: Funcionalidad gestionar aplicaciones

IDENTIFICADOR:	***RP-004***	TÍTULO:	Gestionar clouds
DESCRIPCIÓN:	Los elementos definidos anteriormente son los que nos permiten definir los clouds. Un cloud se definirá en el sistema como un conjunto de máquinas virtuales (componente hardware) encargadas de realizar una serie de tareas (componente software). Por ello, la aplicación deberá permitir la asociación de las máquinas virtuales (en adelante MVs) con las tareas anteriormente definidas formando una nube resultante, que será el objetivo de nuestras simulaciones.		

Tabla 4: Funcionalidad gestionar clouds

IDENTIFICADOR:	***RP-005***	TÍTULO:	Gestionar simulaciones
DESCRIPCIÓN:	La herramienta "iCanCloud" se encarga de realizar las simulaciones de un determinado cloud previamente configurado y obtener los resultados oportunos. Para ello, dicho simulador debe leer previamente una serie de archivos de configuración donde se encuentran los parámetros necesarios para llevar a cabo el cálculo de costes y tiempos de ejecución. Así, la aplicación a desarrollar deberá traducir la información legible que ha sido introducida por el usuario en la interfaz a los ficheros de configuración que el simulador "entiende". Además de esto, la aplicación debe ser capaz de interactuar con el SO para lanzar la simulación que lea estos ficheros generados. Este proceso incluye la configuración y carga de variables de entorno y demás información de importancia que se detallará más adelante.		

Tabla 5: Funcionalidad gestionar simulaciones

IDENTIFICADOR:	***RP-006***	TÍTULO:	Gestionar resultados
DESCRIPCIÓN:	Una vez la aplicación haya sido capaz de generar el entorno deseado por el usuario, traducir a código "legible" por el simulador y ejecutar de forma satisfactoria dicha simulación, se debe interpretar los resultados obtenidos de la simulación. Así, la aplicación deberá leer los ficheros de resultados creados por el simulador, almacenar los datos de interés y tratarlos, de manera que el usuario consiga una descripción detallada de lo ocurrido en el proceso.		

Tabla 6: Funcionalidad gestionar resultados

IDENTIFICADOR:	***RP-007***	TÍTULO:	Gestionar la configuración del usuario
DESCRIPCIÓN:	Cuando el usuario esté utilizando la herramienta para gestionar simulaciones de clouds, irremediablemente tendrá creados una serie de elementos (máquinas virtuales, tareas, etc.) con los que trabajar. El sistema deberá gestionar esta información de forma que facilite al máximo el trabajo del usuario. Por ejemplo, cada vez que el usuario acceda a la aplicación se deberá mostrar la información con la que estaba trabajando la última vez. Por ello, la aplicación deberá implementar funciones como el guardado y recuperación de la configuración, aunque también se podrían implementar otras como la importación y exportación de una configuración, etc.		

Tabla 7: Funcionalidad gestionar configuración usuario

Como producto de la simplificación de las distintas funcionalidades comentadas anteriormente y teniendo como objetivo conseguir un mayor grado de detalle en ellas, surgen una serie de funcionalidades básicas, derivadas de las anteriores, que debe cumplir nuestra herramienta. Estas funcionalidades básicas son las siguientes:

IDENTIFICADOR:	***RB-001***	TÍTULO:	Crear nueva máquina virtual
DESCRIPCIÓN:	La aplicación deberá ser capaz de crear nuevos modelos de máquinas virtuales totalmente configurables por parte del usuario. Así, el usuario podrá crear nuevas instancias de máquinas virtuales identificadas mediante un código definido por el usuario y diferenciadas entre sí por los valores que el propio usuario decida dar a los parámetros. Para mantener la integridad de los datos, la aplicación deberá comprobar que no se genere ninguna máquina virtual que ya haya sido creada anteriormente, es decir, que no existan dos MVs con el mismo código identificador.		

Tabla 8: Funcionalidad crear nueva máquina virtual

IDENTIFICADOR:	***RB-002***	TÍTULO:	Borrar máquina virtual
DESCRIPCIÓN:	El sistema deberá ofrecer al usuario la opción de borrar un modelo de máquina virtual que haya sido anteriormente creada. La aplicación deberá pedir confirmación al usuario sobre el borrado y en el caso de que la contestación sea afirmativa, deberá borrar la información del sistema. La máquina a borrar puede estar incluida en algún cloud previamente creado. En el caso, de que se decida borrar una máquina virtual que pertenezca a algún cloud, éste se actualizará de forma que se eliminará la relación de pertenencia entre ambas entidades.		

Tabla 9: Funcionalidad borrar máquina virtual

IDENTIFICADOR:	***RB-003***	TÍTULO:	Modificar máquina virtual
DESCRIPCIÓN:	La aplicación deberá permitir modificar un modelo de máquina virtual creada previamente. Así, el usuario tendrá la posibilidad de modificar los parámetros establecidos anteriormente por los que el usuario desee, sin necesidad de borrar la máquina y volver a crearla con los nuevos datos.		

Tabla 10: Funcionalidad modificar máquina virtual

IDENTIFICADOR:	***RB-004***	TÍTULO:	Borrar todas las máquinas virtuales
DESCRIPCIÓN:	La aplicación deberá ofrecer la posibilidad de borrar todos los modelos de máquinas virtuales almacenadas en el sistema. Esta operación también requerirá la confirmación del usuario. Según lo comentado anteriormente, se borrarán todas las dependencias con respecto a los clouds en los que estuvieran incluidas estas máquinas, pero no se borrarán dichos clouds, sino que se modificarán adecuadamente.		

Tabla 11: Funcionalidad borrar todas las máquinas virtuales

IDENTIFICADOR:	***RB-005***	TÍTULO:	Crear nueva tarea
DESCRIPCIÓN:	Como se comentó anteriormente, las tareas conforman el componente software de la nube. El sistema deberá ofrecer la posibilidad de crear nuevas tareas o trabajos de usuario. Así, el usuario podrá seleccionar las aplicaciones que quieren que se simulen en el cloud, el número y tipo de máquinas que se emplearán para la simulación o el número de simulaciones que se lanzarán. Además, al igual que ocurría con las MVs, no se permitirá crear una tarea que haya sido anteriormente generada, por lo que el sistema deberá devolver un mensaje de error que avise al usuario.		

Tabla 12: Funcionalidad crear nueva tarea

IDENTIFICADOR:	***RB-006***	TÍTULO:	Borrar tarea
DESCRIPCIÓN:	La aplicación deberá proporcionar la funcionalidad necesaria para eliminar un modelo de tarea que haya sido previamente definida en el sistema. Al igual que ocurría en el caso de las MVs, la ejecución de esta acción deberá devolver un mensaje en el que se solicite la confirmación del usuario en cuestión. En el caso de que la respuesta sea afirmativa, se eliminará la tarea de la base de datos del sistema, así como su posible pertenencia a cualquier de los clouds previamente definidos. Es decir, todos los clouds en los que se ejecutaran estas tareas perderían irremediablemente dicha información.		

Tabla 13: Funcionalidad borrar tarea

IDENTIFICADOR:	***RB-007***	TÍTULO:	Modificar tarea
DESCRIPCIÓN:	El sistema debe permitir la modificación de modelos de tareas previamente definidos. De esta forma, el usuario tendrá la posibilidad de añadir a ejecución nuevas aplicaciones, cambiar la cantidad de máquinas sobre las que se lanzará una aplicación, etc., todo esto sin la necesidad de eliminar la tarea y volver a crear otra con los datos nuevos.		

Tabla 14: Funcionalidad modificar tarea

IDENTIFICADOR:	***RB-008***	**TÍTULO:**	Borrar todas las tareas
DESCRIPCIÓN:	La aplicación deberá permitir el borrado de todas las tareas que hayan sido almacenadas en la base de datos del sistema. Como en todos los procesos de borrado vistos hasta ahora se pedirá la confirmación del usuario y se eliminará toda la información relacionada en caso afirmativo. Al igual que antes, se actualizarán todos los clouds definidos en el sistema con motivo del borrado de todas estas tareas.		

Tabla 15: Funcionalidad borrar todas las tareas

IDENTIFICADOR:	***RB-009***	**TÍTULO:**	Crear nueva aplicación
DESCRIPCIÓN:	Los modelos de aplicaciones representan el software que se ejecutará sobre las máquinas virtuales. El sistema deberá permitir crear nuevos modelos de aplicaciones a partir de un tipo de aplicación dado. Es decir, el sistema deberá leer de forma dinámica los tipos de aplicaciones definidos (una aplicación para medir distancias, otra para calcular velocidades,…) y permitir al usuario crear una aplicación del tipo que desee. Además, cada aplicación creada por el usuario podrá ser única ya que se ofrecerá la posibilidad de definir el valor de una serie de parámetros (por ejemplo, la posición de dos puntos para el primer caso anterior). Si bien ya se ha comentado que se permitirá añadir nuevos tipos de aplicaciones de forma dinámica (es decir, los tipos no estarán limitados en la aplicación), en el sistema vienen por defecto tres tipos de aplicaciones que el usuario podrá personalizar e incluir en sus trabajos para el cloud. Estos tipos de aplicaciones son: ○ BasicApplication ○ DummyApplication ○ LocalApplication Cada una de estas aplicaciones tiene sus parámetros propios, por lo que la aplicación a desarrollar deberá solicitar los parámetros adecuados dependiendo del tipo de aplicación escogida por el usuario durante el proceso de alta. Además, al igual que ocurría con las MVs, no se permitirá crear una tarea		

	que haya sido anteriormente generada, por lo que el sistema deberá devolver un mensaje de error que avise al usuario.

Tabla 16: Funcionalidad crear nueva aplicación

IDENTIFICADOR:	***RB-0010***	TÍTULO:	Borrar aplicación
DESCRIPCIÓN:	El sistema deberá ofrecer la funcionalidad necesaria para eliminar un modelo de aplicación que haya sido previamente definida en el sistema. Al igual que ocurría en el caso de las MVs, la ejecución de esta acción deberá devolver un mensaje en el que se solicite la confirmación del usuario en cuestión. En el caso de que la respuesta sea afirmativa, se eliminará la aplicación de la base de datos del sistema, así como su posible pertenencia a cualquiera de las tareas o trabajos previamente definidos.		

Tabla 17: Funcionalidad borrar aplicación

IDENTIFICADOR:	***RB-011***	TÍTULO:	Modificar aplicación
DESCRIPCIÓN:	El sistema debe permitir la modificación de modelos de aplicaciones previamente definidos. Eso sí, la aplicación permitirá esta modificación siempre y cuando el usuario decida modificar los valores de los parámetros de una tarea “de un determinado tipo”. Es decir, no se permitirá modificar el tipo de aplicación asociado. Esto se ha decidido así ya que realmente este “tipo” es el parámetro que identifica una aplicación de otra y no tiene sentido este cambio. Imaginemos que existe un tipo de aplicación que se identifica por medir el tiempo empleado por un móvil en un trayecto determinado. El usuario puede, en algún momento determinado, modificar datos como la distancia de ese trayecto, la velocidad del móvil, etc., pero nunca va a pensar en transformar esa aplicación en otra basada en, por ejemplo, mediciones de temperatura. Así, se define esta acción como la modificación de tareas de un mismo tipo o dentro de un mismo ámbito.		

Tabla 18: Funcionalidad modificar aplicación

IDENTIFICADOR:	***RB-012***	TÍTULO:	Borrar todas las aplicaciones
DESCRIPCIÓN:	Se ofrecerá al usuario la opción de borrar todas las aplicaciones que hayan sido almacenadas en la base de datos del sistema. Como en todos los procesos de borrado vistos hasta ahora se pedirá la confirmación del usuario y se eliminará toda la información relacionada en caso afirmativo. Al igual que antes, se actualizarán todas las tareas definidas en el sistema con motivo del borrado de todas estas aplicaciones.		

Tabla 19: Funcionalidad borrar todas las aplicaciones

IDENTIFICADOR:	***RB-013***	TÍTULO:	Crear nuevo cloud
DESCRIPCIÓN:	El objetivo principal de la herramienta a desarrollar consiste en la gestión entornos cloud para su posterior simulación. De ello, se deduce que el proceso de creación de un cloud sea una de las acciones fundamentales de la herramienta, a la vez que una de las más complejas. A la hora de crear un nuevo cloud se configurara, por un lado la infraestructura ofrecida por el proveedor (tipo y numero de MVs) y por otro lado las tareas de usuario que se desean ejecutar. Además de las comprobaciones básicas (que el cloud no exista previamente) el sistema deberá comprobar que las máquinas que se solicitan para las tareas asignadas están disponibles en la estructura del cloud (que haya suficientes máquinas de cada tipo para lo que soliciten las tareas).		

Tabla 20: Funcionalidad crear nuevo cloud

IDENTIFICADOR:	***RB-014***	TÍTULO:	Borrar cloud
DESCRIPCIÓN:	Se ofrecerá al usuario la posibilidad de borrar un modelo de cloud previamente incluido en el sistema. Esta acción requerirá la confirmación del usuario, al igual que ocurre con todas las tareas de borrado incluidas en la aplicación.		

Tabla 21: Funcionalidad borrar cloud

IDENTIFICADOR:	***RB-015***	TÍTULO:	Modificar cloud
DESCRIPCIÓN:	La aplicación permitirá modificar los parámetros de un cloud previamente creado. Así, el usuario podrá cambiar el numero de MVs ofrecidas por el proveedor o los trabajos de usuario que se ejecutaran en el, sin necesidad de borrar y volver a generar la información en el sistema. Al igual que ocurría con la creación de clouds nuevos, tras modificar adecuadamente uno previamente definido, se deberá comprobar que se ofrecen suficientes máquinas de cada tipo como para satisfacer la demanda de máquinas que se requieran en las tareas de usuario definidas en el cloud.		

Tabla 22: Funcionalidad modificar cloud

IDENTIFICADOR:	***RB-016***	TÍTULO:	Borrar todos los clouds
DESCRIPCIÓN:	Se permitirá al usuario borrar todos los modelos de clouds del sistema, para lo que se solicitará la confirmación del usuario previamente. El hecho de que se borren todos los clouds del sistema y que el cloud sea la entidad fundamental de la herramienta, no quiere decir que se borren los datos asociados a los recursos incluidos en los clouds. Es decir, el borrado, ya sea de uno o de todos los clouds del sistema, acarrea la lógica eliminación de las relaciones entre estos y los recursos (máquinas virtuales y tareas) pero no habrá ningún cambio en la gestión de estos últimos.		

Tabla 23: Funcionalidad borrar todos los clouds

IDENTIFICADOR:	***RB-017***	TÍTULO:	Generar ficheros de configuración
DESCRIPCIÓN:	La aplicación permitirá al usuario escoger un determinado cloud previamente creado y lanzar una simulación a partir del mismo. Para ello, la aplicación deberá traducir la información encapsulada en el cloud elegido y generar los ficheros de configuración necesarios que requiere el simulador para su ejecución.		

Tabla 24: Funcionalidad generar ficheros de configuración

IDENTIFICADOR:	*RB-018*	TÍTULO:	Lanzar simulación
DESCRIPCIÓN:	Esta labor va estrechamente ligada a la acción anterior ya que ambas se ejecutarán cuando el usuario decida simular un entorno de cloud concreto. Así, la aplicación deberá leer los ficheros de configuración generados con anterioridad y comunicarse con el simulador para así lanzar una ejecución, comunicándole a éste último tanto los datos leídos como otros datos de configuración que deberá gestionar el propio sistema.		

Tabla 25: Funcionalidad lanzar simulación

IDENTIFICADOR:	*RB-019*	TÍTULO:	Monitorizar simulación
DESCRIPCIÓN:	La aplicación deberá mostrar en todo momento al usuario la información en primicia sobre el estado de la ejecución. El simulador devuelve durante su ciclo de ejecución una serie de logs que indican el estado del proceso. La aplicación deberá leer esta información devuelta por el simulador (información de logs) y mostrársela el usuario a través de la interfaz gráfica, mientras el simulador continúa con su trabajo.		

Tabla 26: Funcionalidad monitorizar simulación

IDENTIFICADOR:	*RB-020*	TÍTULO:	Interpretar resultados
DESCRIPCIÓN:	La aplicación deberá ofrecer la funcionalidad de acceder a los resultados obtenidos tras la realización de la simulación (el simulador crea unos ficheros de resultados cuando termina su ejecución) y almacenarlos en el sistema. Una vez almacenados estos datos deben poder ser tratados para su posterior presentación al usuario (obtención de conclusiones, comparativas, etc.)		

Tabla 27: Funcionalidad interpretar resultados

IDENTIFICADOR:	***RB-021***	TÍTULO:	Generar gráficas
DESCRIPCIÓN:	La aplicación deberá ser capaz de trabajar con los datos obtenidos como resultado de las simulaciones lanzadas y tratarlos adecuadamente para conseguir un conjunto de conclusiones. A partir de éstas, la aplicación deberá generar una serie de gráficas, que reflejen las estadísticas más relevantes acerca de la simulación realizada, como pueden ser variaciones de tiempos de ejecución, variaciones de costes, tasas acumuladas, etc.		

Tabla 28: Funcionalidad generar gráficas

IDENTIFICADOR:	***RB-022***	TÍTULO:	Generar informe
DESCRIPCIÓN:	Se ofrecerá al usuario la posibilidad de generar documentos de informe que recojan la actividad relacionada con la simulación realizada. Se podrá generar un informe por simulación (o lo que es lo mismo, por cloud). Este informe utilizará la información de salida del simulador debidamente tratada. Con este método, el usuario podrá tener constancia a posteriori de lo ocurrido en la simulación y le proporcionará la ayuda necesaria para la toma de decisiones.		

Tabla 29: Funcionalidad generar informe

IDENTIFICADOR:	***RB-023***	TÍTULO:	Guardar configuración
DESCRIPCIÓN:	La aplicación deberá ofrecer al usuario la opción de guardar los datos que está manejando en un momento determinado (su configuración personal). Así, se guardarán todos los datos relativos a las máquinas virtuales, las tareas, las aplicaciones y los clouds generados en esa sesión y/o en anteriores.		

Tabla 30: Funcionalidad guardar configuración

IDENTIFICADOR:	*RB-024*	TÍTULO:	Cargar configuración
DESCRIPCIÓN:	Si antes hablábamos de que la aplicación debe permitir al usuario guardar en todo momento su configuración personal, se puede deducir que otro requisito imprescindible consiste en el hecho de que el sistema deba proporcionar un mecanismo para la recuperación de la información almacenada. Así, cada vez que el usuario arranque la aplicación, ésta deberá comprobar qué información guardó el usuario antes de salir y cargar dicha información de nuevo en la herramienta. Como se puede deducir, esta acción será totalmente transparente al usuario y la llevará a cabo la herramienta de manera automática.		

Tabla 31: Funcionalidad cargar configuración

3.6 Casos de uso

Con la información de las funcionalidades a implementar, se puede realizar un diagrama general de casos de uso para representar la interacción del sistema con el usuario (o con los actores, como se denominan en este tipo de diagramas).

Con el fin de conseguir una mejor visión de la información, se ha decidido dividir el esquema principal en una serie de subesquemas, cada uno de los cuáles muestra una parte de la funcionalidad antes descrita. Así, primero se muestran los casos de uso principales y posteriormente se describe cada uno de ellos junto con los casos derivados del mismo.

La Figura 13 muestra el esquema principal de casos de uso del sistema. En él se muestran las posibles acciones entre los usuarios (el actor) y la aplicación. La funcionalidad principal definida a lo largo de este documento se divide en siete casos de uso principales: gestión de clouds, gestión de máquinas virtuales, gestión de tareas, gestión de aplicaciones, gestión de simulaciones, gestión de resultados y gestión de la configuración del usuario.

Cada uno de estos casos de uso individuales se divide a su vez en una serie de casos de uso básicos.

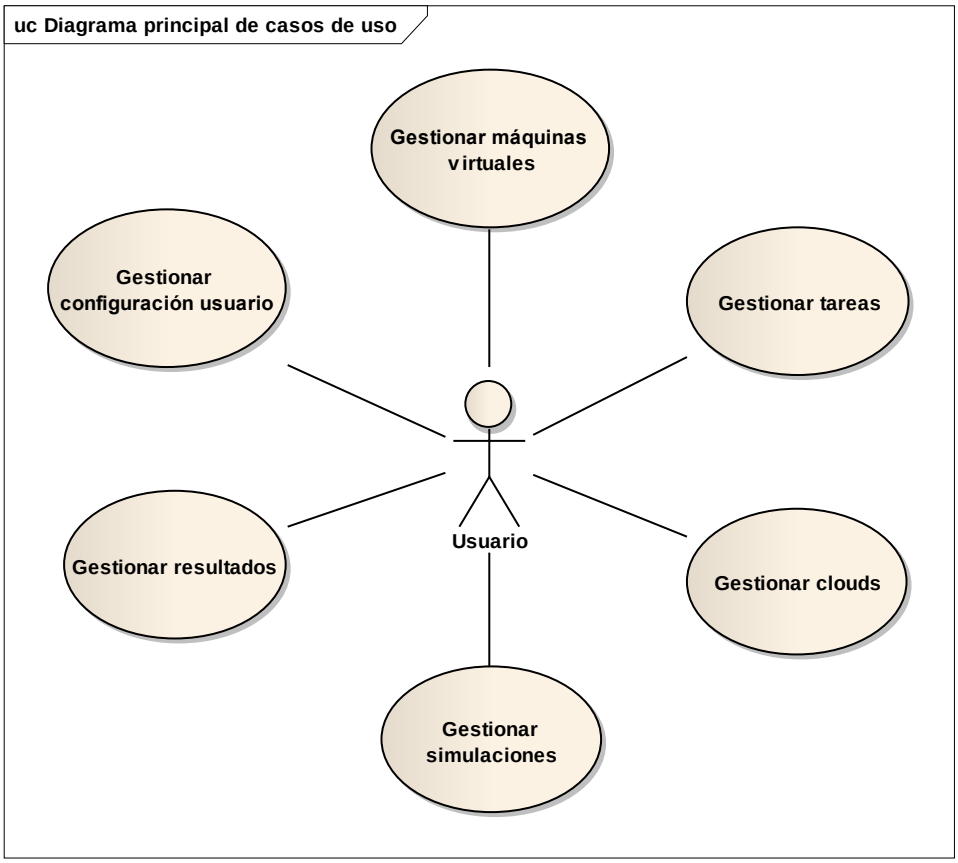

Figura 13: Casos de uso principales

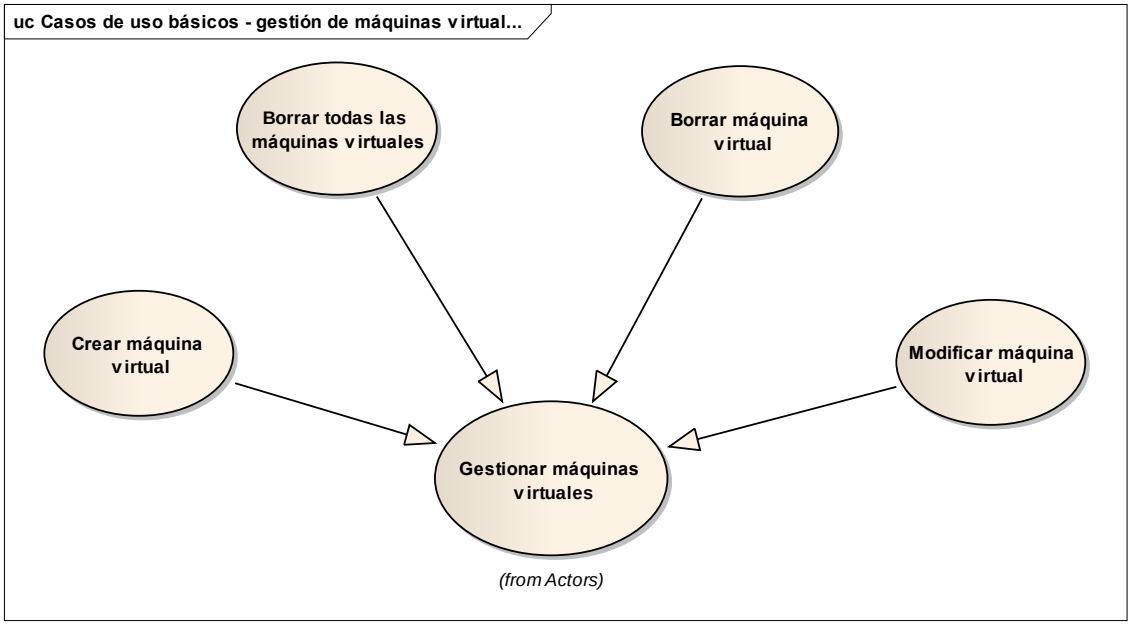

Figura 14: Casos de uso derivados de la gestión de las VM

En la Figura 14 se muestran los casos de uso derivados de la gestión de máquinas virtuales. Se puede observar cómo esta funcionalidad se ha dividido en otras cuatro dependiendo de la acción a realizar en uno u otro caso.

La Figura 15 que aparece a continuación muestra los casos de uso derivados de la gestión de tareas en el sistema.

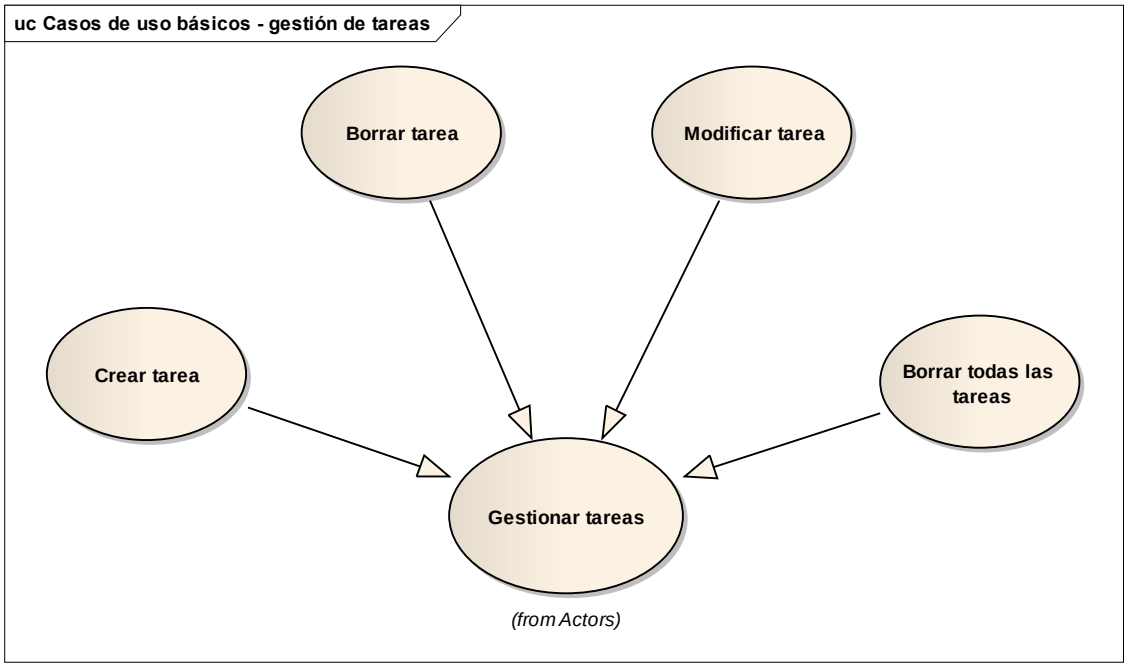

Figura 15: Casos de uso derivados de la gestión de tareas

De nuevo, esta funcionalidad se ha dividido en otras cuatro de manera similar al caso de las máquinas virtuales.

Por su parte, la Figura 16 muestra los casos derivados de la gestión de clouds. Al igual que ha ocurrido con las MVs y las tareas, esta funcionalidad ha sido dividida en cuatro casos de uso, dependiendo de la acción a realizar por cada uno de ellos.

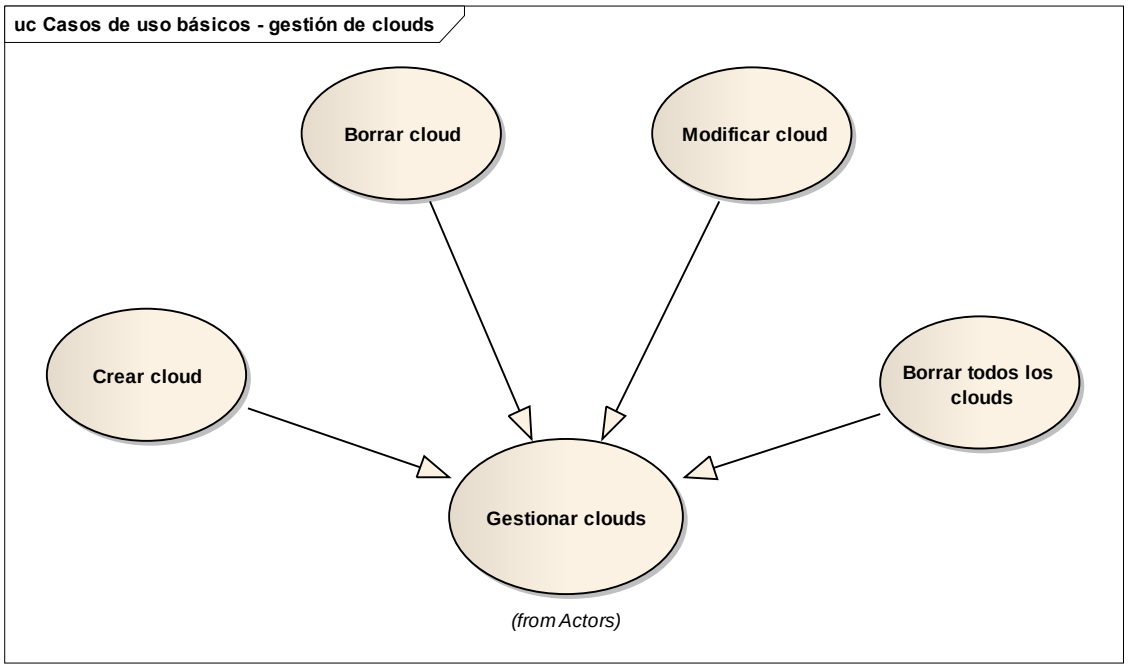

Figura 16: Casos de uso derivados de la gestión de clouds

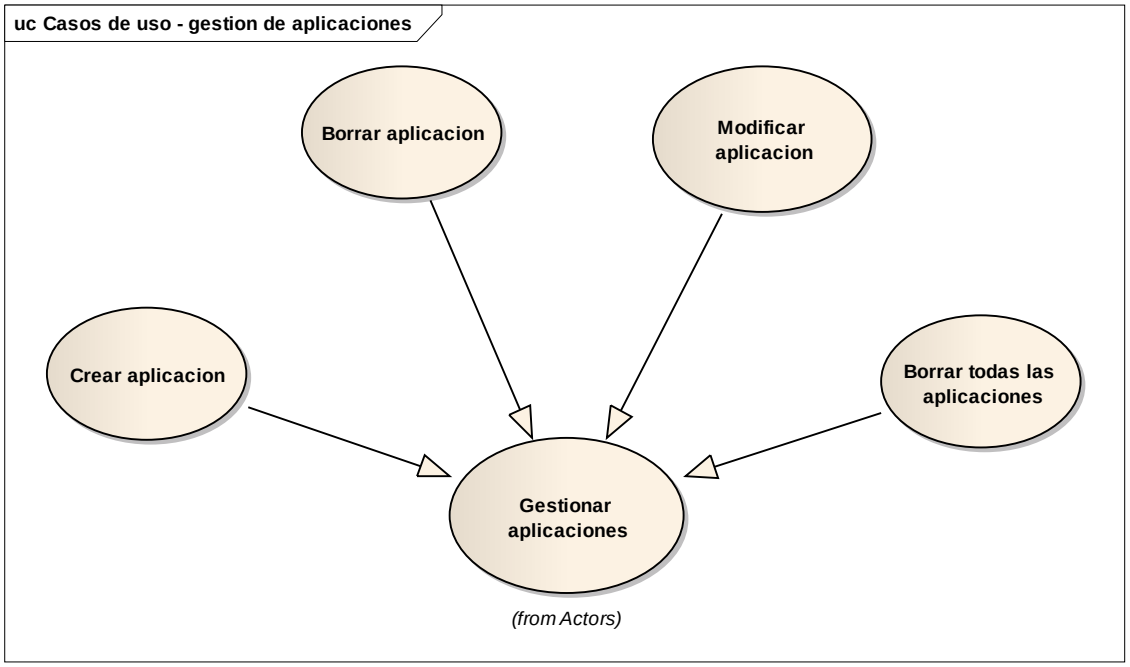

Figura 17: Casos de uso derivados de la gestión de aplicaciones

La Figura 17 muestra los casos de uso derivados de la gestión de aplicaciones en el sistema. Al igual que ocurría con los elementos anteriores, el caso de uso principal se divide en cuatro nuevos casos de uso, dependiendo de la acción a realizar por cada uno de ellos.

En lo que respecta a la gestión de las simulaciones, a continuación se muestra un esquema de su división:

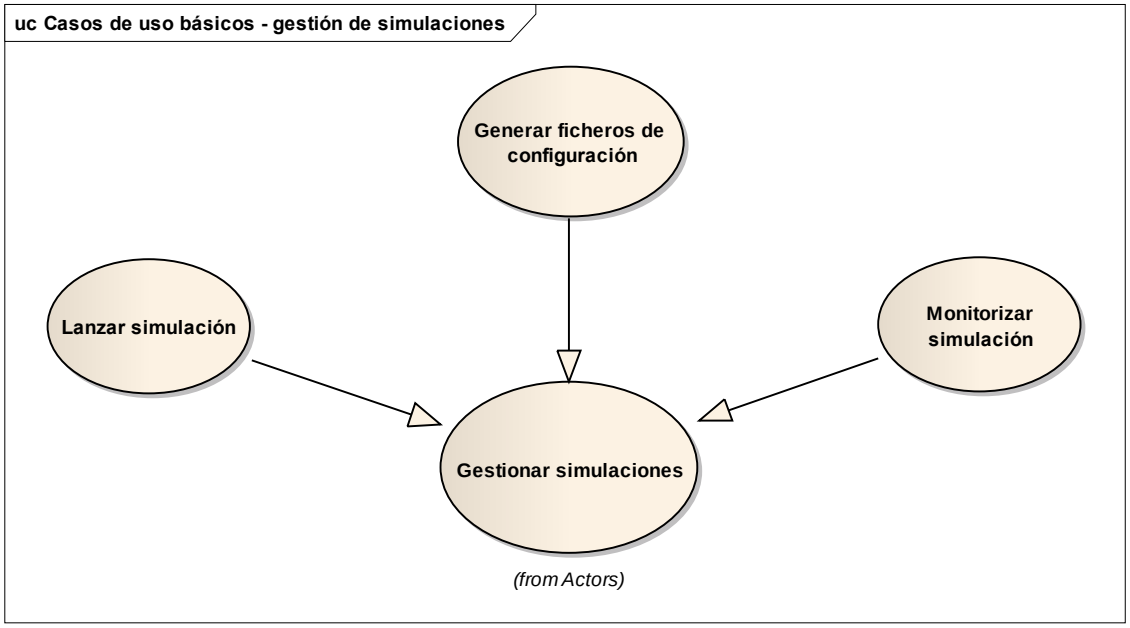

Figura 18: Casos de uso derivados de la gestión de simulaciones

En la anterior Figura 18, se muestran los casos de uso derivados de la gestión de simulaciones. En este caso aparecen tres nuevos casos de uso muy vinculados entre sí, ya que todos tendrán su punto de inicio en una situación determinada aunque manejarán distinto tipo de información de usuario.

Por su parte, la Figura 19 representa los casos de uso derivados de la gestión de resultados, los cuales se dividen en tres casos distintos.

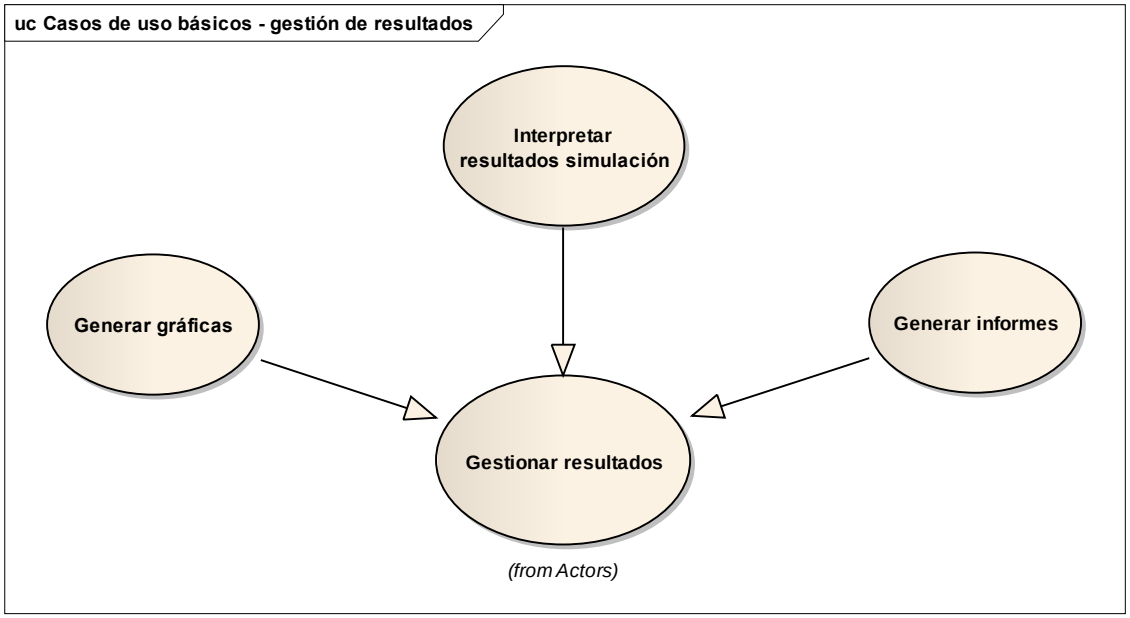

Figura 19: Casos de uso derivados de la gestión de resultados

Por último, se recogen en la Figura 20 los casos de uso derivados de la gestión de la configuración:

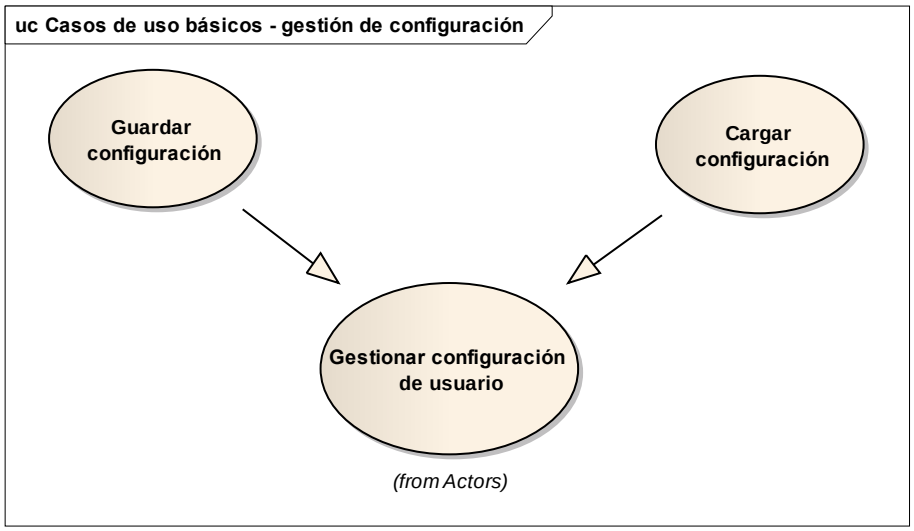

Figura 20: Casos de uso derivados de la gestión de la configuración

Capítulo

4

4. Diseño del sistema

En el apartado anterior se describía el sistema a desarrollar de manera global, prestando especial atención a la definición de la funcionalidad que debía cubrir. En éste, nos centraremos en describir el software a desarrollar. Para ello, se comenzará definiendo cual es la arquitectura utilizada para el desarrollo del software, para después generar un modelo lógico a partir de la información obtenida en el análisis y teniendo en cuenta la arquitectura escogida.

Después se describirán cada uno de los componentes de este modelo y se definirán las interacciones entre ellos en los denominados "diagramas de secuencia". Para finalizar este capítulo se detallarán algunas de las funciones principales que intervienen en el sistema y se comentará la sintaxis de los ficheros generados o leídos por la aplicación.

4.1 Detalle de la arquitectura: Modelo Vista Controlador

El patrón Modelo Vista Controlador, también denominado MVC, es un patrón de arquitectura de software que separa los datos de la aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos.

Este modelo es generalmente utilizado en aplicaciones con interfaces sofisticadas o en aquellas donde se desea abstraer la parte de interacción con el usuario de la parte de gestión de los datos propios del sistema.

En el caso de nuestra aplicación, es más probable que surjan cambios en la lógica de la interfaz del usuario que en el resto de elementos. La interacción con el simulador y los tipos de datos con los que vamos a trabajar (máquinas virtuales, tareas, clouds), así como la lógica del negocio, son elementos que previsiblemente no van a variar en nuestro sistema. Si elegimos una arquitectura de software donde estos componentes no estén claramente diferenciados, por ejemplo, mezclando componentes de la interfaz y la lógica de la aplicación, podría darse el

caso de que un determinado cambio en la interfaz nos suponga un arduo trabajo con los componentes de la lógica.

Por ello, se ha decidido trabajar con una arquitectura que separe perfectamente la vista, del modelo de datos y de la lógica de la aplicación: el patrón Modelo Vista Controlador.

La Figura 21 representa esta metodología de programación en forma de pirámide. Como se puede observar, esta arquitectura está perfectamente diferenciada en tres secciones principales, que se describen a continuación:

- **Modelo**: encapsula los datos y las funcionalidades. El modelo es totalmente independiente de cualquier representación de salida y/o comportamiento de entrada.
- **Vista**: muestra la información del modelo al usuario. Pueden existir varias vistas del modelo. Cada vista tiene asociado un componente controlador.
- **Controlador**: gestiona las entradas del usuario. Estas entradas se implementan generalmente como eventos. Estos eventos son traducidos a solicitudes de servicio para el modelo o la vista.

Como se ha comentado, en esta arquitectura pueden existir varias vistas y varios controladores (uno para cada vista) pero el modelo de datos siempre es fijo, por ello se suele representar en la base de la pirámide.

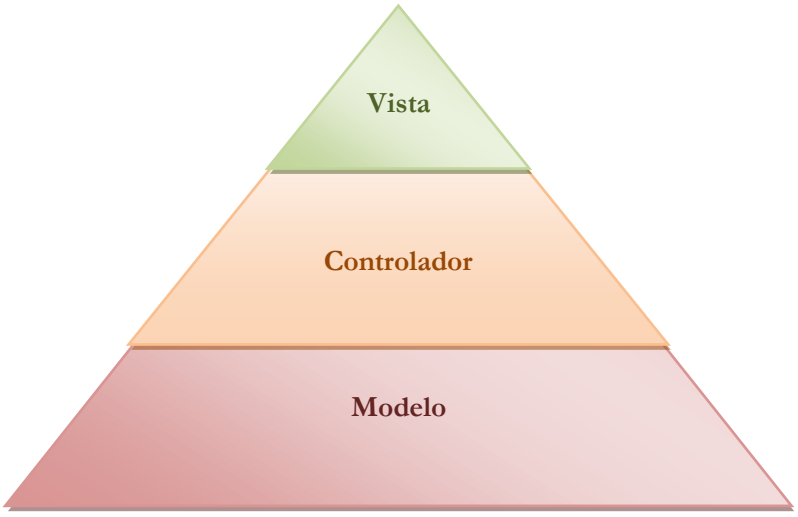

Figura 21: Arquitectura por capas MVC

4.2 Modelo lógico

Una vez que se han definido los requisitos a cubrir, se ha descrito la funcionalidad a implementar y se ha elegido la arquitectura en la que se va a apoyar nuestra aplicación, ya se puede diseñar un modelo lógico de datos.

El modelo representado en la Figura 22, contiene los principales conceptos incluidos en la especificación de requisitos de usuario además de las estructuras de datos necesarias para asegurar el correcto funcionamiento del sistema.

En dicho modelo, se aprecian perfectamente las distintas capas que forman una arquitectura Modelo Vista Controlador. En la capa superior aparecen los componentes que formarán parte de la Vista del diseño (zona verde) y que tienen como cometido principal la interacción con el usuario de la aplicación, tanto para la recogida de peticiones como para la muestra de resultados.

En la capa intermedia se recogen los componentes que conformarán la parte del Controlador (zona naranja), que es el encargado de gestionar las peticiones del cliente procedentes de la vista (mediante la conexión GUI - RequestManager) y su reenvío al modelo de datos para su tratamiento y viceversa.

En la capa inferior (zona roja) se recogen los elementos que forman parte del Modelo del sistema. Estos componentes corresponden con los conceptos incluidos en la especificación de requisitos y forman el núcleo invariable de la aplicación. Esta capa encapsula los datos y la funcionalidad principal del sistema.

Por lo tanto, si colocáramos a un hipotético usuario de la aplicación en la parte superior de esta torre de componentes comprobaríamos que se cumple el flujo normal de control de cualquier arquitectura MVC.

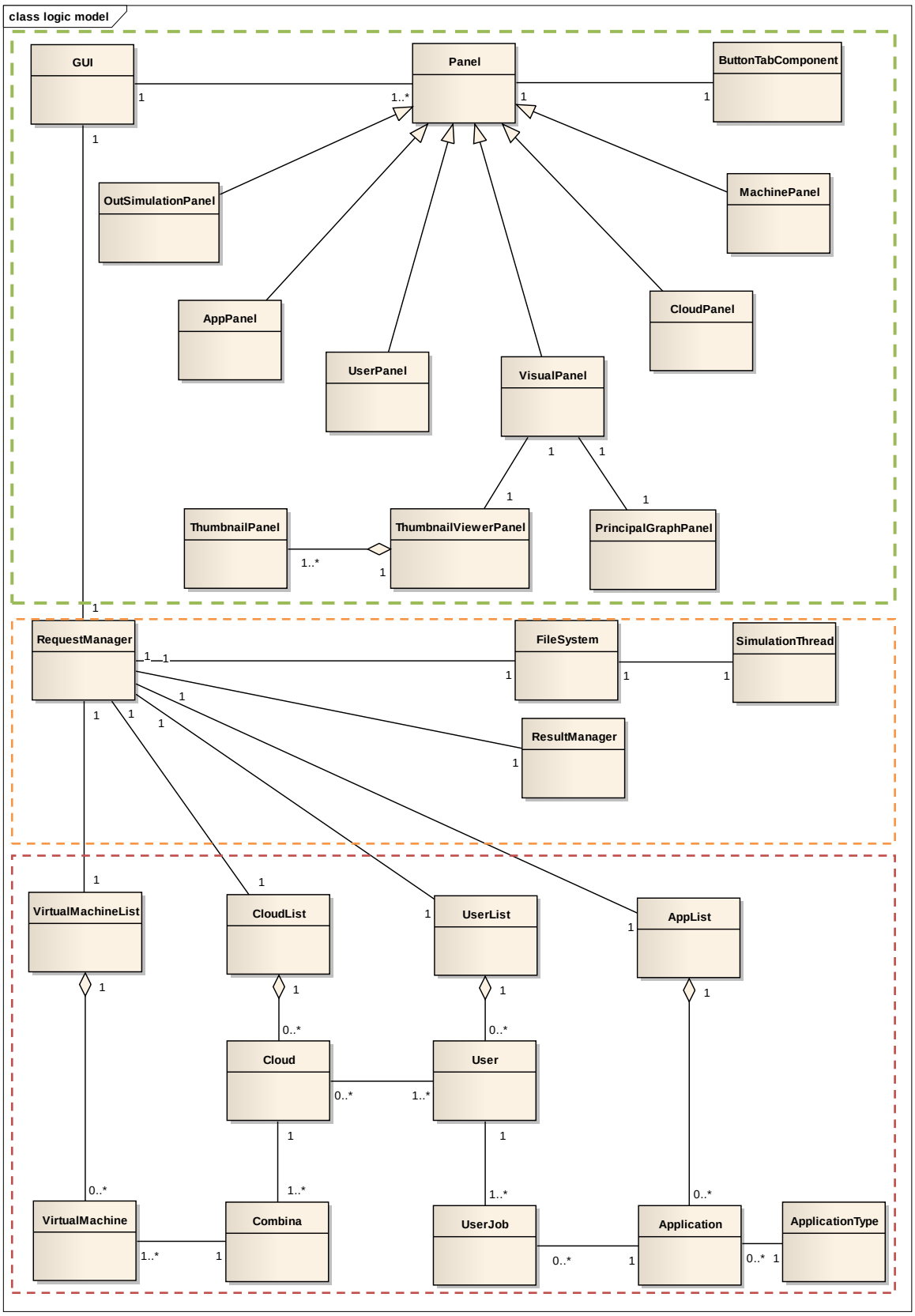

Figura 22: Modelo lógico del sistema

Este flujo es el siguiente:

1. El usuario realiza una acción en la interfaz (en este caso, sobre el componente GUI o alguno de sus componentes derivados).
2. El controlador trata el evento de entrada (el componente GUI se lo comunica a RequestManager y este lo trata). Para poder hacerlo debe haber sido registrado previamente (el componente debe conocer de qué acción se trata y cómo tiene que actuar).
3. El controlador notifica al modelo la acción del usuario, lo que puede implicar un cambio del estado del modelo (si se trata de una actualización y no de una simple consulta).
4. Se genera una nueva vista. La vista o el controlador toman los datos del modelo (en nuestro caso, se ha decidido centralizar este comportamiento en el componente RequestManager).
5. La interfaz de usuario espera una nueva interacción del usuario, que provocará que comience otro nuevo ciclo

4.3 Diagramas de secuencia

En este apartado se muestran los diagramas de secuencia asociados a los casos de uso vistos en apartados anteriores. En estos diagramas se refleja la secuencia de acciones que se originan entre los componentes del diseño descritos en el apartado anterior.

La Figura 23 muestra un escenario en el que un usuario de la aplicación crea una nueva máquina virtual. Para ello, el usuario mandará un mensaje a la interfaz diciendo que se dispone a crear una nueva máquina, por lo que la aplicación habilitará una nueva pantalla para tal propósito. Después de introducir convenientemente los datos asociados a la nueva máquina a crear, el usuario enviará el mensaje **AddMachineToList(data)** que incluirá todos los parámetros asociados a la nueva máquina a crear y entre los que destaca el nombre con el que se identificará a la máquina dentro del sistema. La clase RequestManager, que es la encargada de gestionar las peticiones del cliente, comprobará si ha sido creada anteriormente alguna máquina con ese nombre mediante **GetVirtualMachineByName(name)** y en caso negativo se creará una instancia de la nueva máquina virtual y del nuevo tipo de máquina (este último se utiliza en la gestión de los clouds).

Crear máquina virtual

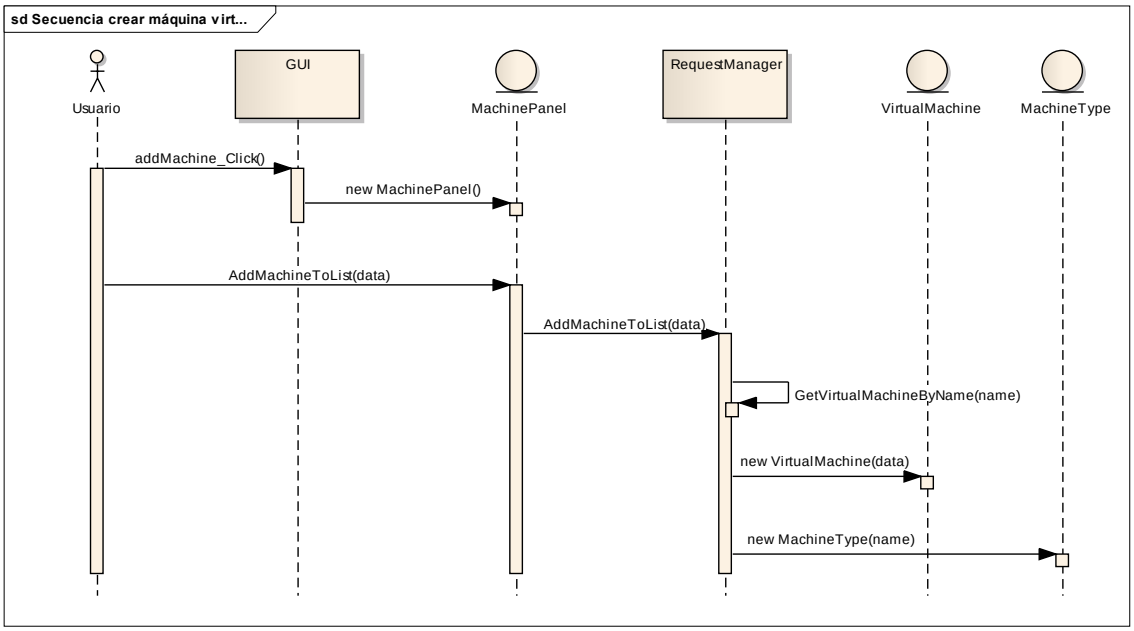

Figura 23: Secuencia crear máquina virtual

Por su parte, la Figura 24 representa el caso en el que un usuario de la aplicación decide modificar el valor de los parámetros asociados a una máquina virtual. Inicialmente el usuario interactúa con la aplicación indicando el tipo de acción que quiere llevar a cabo, en este caso, modificar los parámetros de la máquina virtual cuyo identificador es "name". En este momento se realizan dos acciones: por un lado la interfaz devuelve una nueva pantalla o área de trabajo al usuario, y por otro lado se solicitan los datos de la máquina a modificar mediante el envío del mensaje **GetInfoVirtualMachineByName(name)**. Después de este proceso, se muestra al usuario una nueva pantalla en la que se han cargado los valores asociados a la máquina deseada.

Una vez que el usuario tiene todos los datos de la máquina deseada en pantalla, puede interactuar con la interfaz para modificarlos. Así, cuando se hayan modificado los valores oportunos, se envía un mensaje **EditVirtualMachine(data)** con los nuevos datos a guardar. Finalmente, se muestra al usuario un mensaje de confirmación de que el proceso ha finalizado correctamente.

Modificar máquina virtual

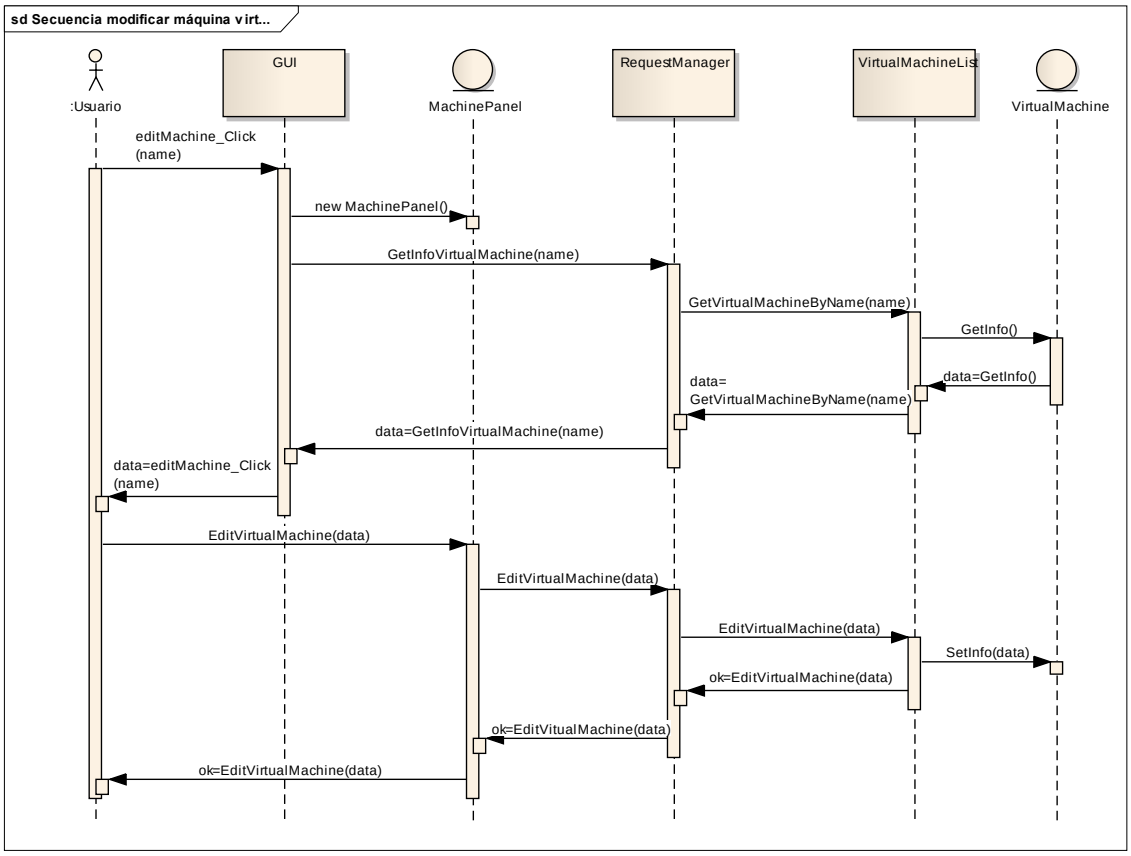

Figura 24: Secuencia modificar máquina virtual

El esquema representado en la Figura 25 muestra la secuencia de acciones asociada a la funcionalidad de borrar una máquina virtual del sistema. El usuario envía un mensaje a la aplicación para comunicar que desea eliminar una máquina determinada (enviando para ello el nombre identificativo de la máquina). Esto provoca que la aplicación devuelva un mensaje de petición de confirmación al mismo. Si el usuario confirma este borrado, se realiza una llamada al método **DeleteMachine(name)** del gestor de peticiones, que realiza a su vez una serie de llamadas hasta llegar a la eliminación de la máquina deseada y el envío de un mensaje de notificación por parte del sistema.

Borrar máquina virtual

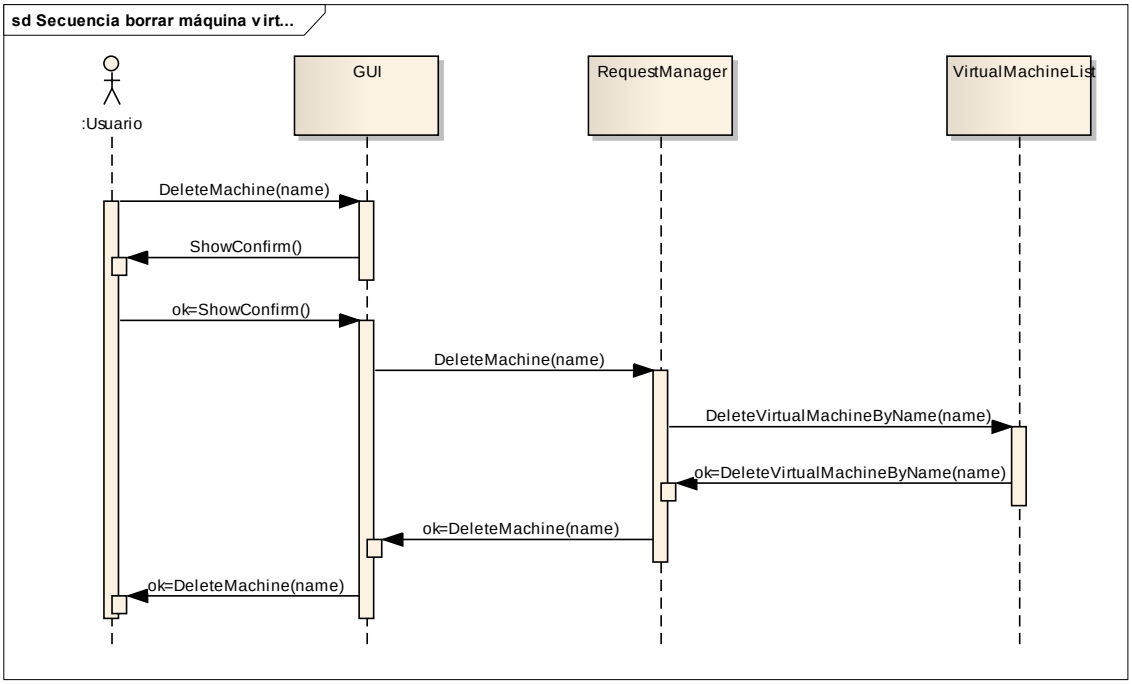

Figura 25: Secuencia borrar máquina virtual

Borrar todas las máquinas virtuales

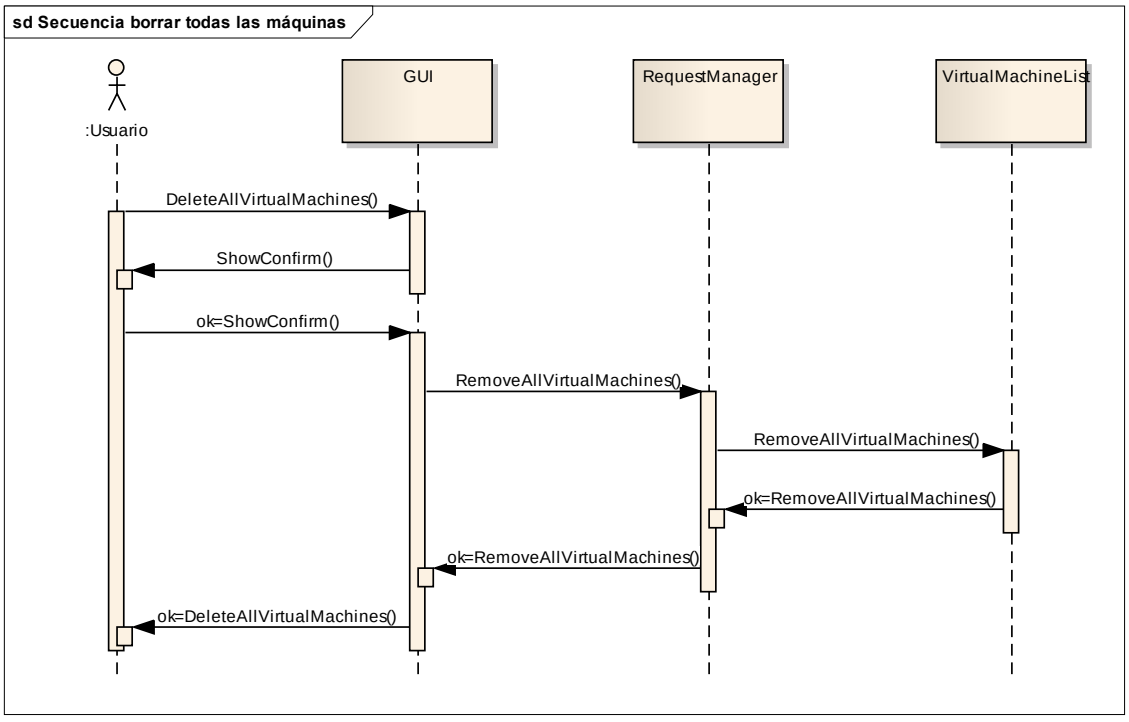

Figura 26: Secuencia borrar todas las máquinas virtuales

La Figura 26 muestra el caso en el que el usuario se dispone a borrar todas las máquinas virtuales almacenadas. Se envía, por parte del usuario, un mensaje **DeleteAllVirtualMachines()** que requiere confirmación. En el caso de que se confirme esta solicitud de borrado, se enviarán una serie de mensajes que finalizarán con la eliminación de todas las máquinas del sistema y de los tipos de máquinas almacenados, junto con la notificación del proceso al usuario.

Crear aplicación

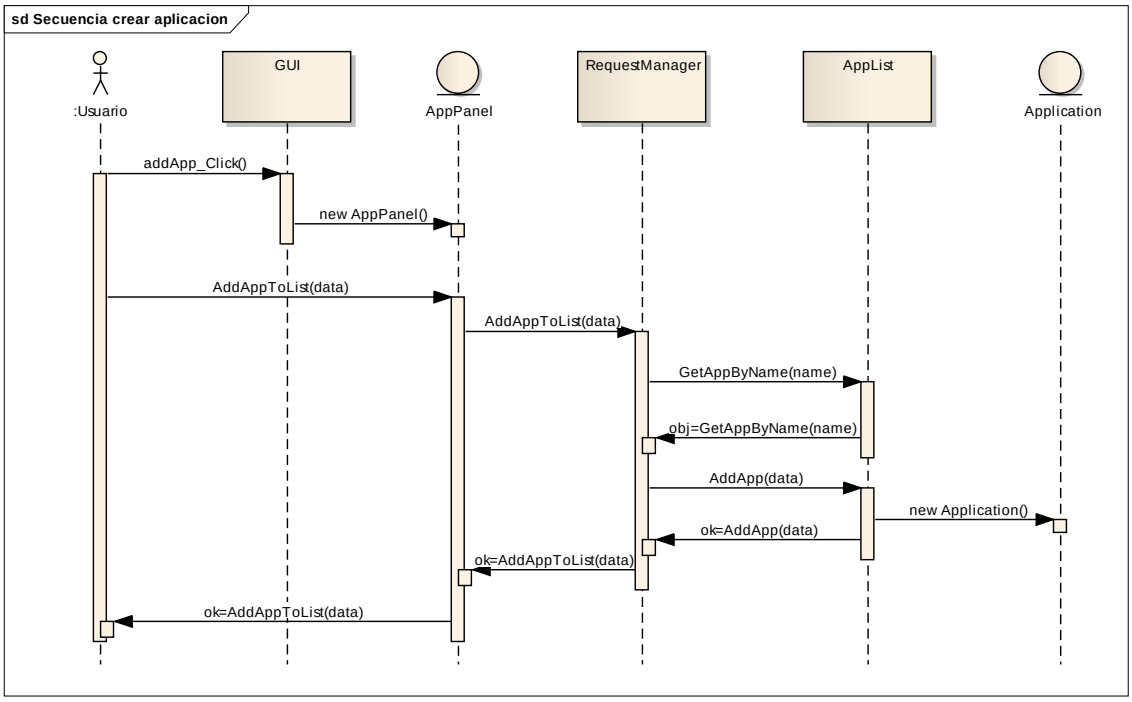

Figura 27: Secuencia crear aplicación

En la Figura 27 aparece reflejado el caso en el que el usuario de la aplicación crea una nueva aplicación en el sistema. Para ello, dicho usuario interactuará con la interfaz para obtener una nueva pantalla en donde registrar los datos propios de la nueva aplicación. Una vez que hayan sido insertados todos los datos de la misma (incluyendo el tipo de aplicación que ha elegido) el usuario deberá aceptar el proceso, por lo que se realiza una llamada al método **AddAppToList(data)**, el cual pertenece al gestor de peticiones. En este punto, el propio gestor se encargará de comprobar la consistencia de los datos respecto a la nueva tarea, por lo que se realiza una llamada al método **GetAppByName(name)** de la clase *AppList* con el nombre de la nueva tarea que se desea añadir. Si se encuentra alguna ocurrencia significa que se ha creado anteriormente una aplicación con ese nombre y se devolverá un mensaje de error al usuario.

En caso contrario, se generará la nueva aplicación y se devolverá el correspondiente mensaje de notificación.

Modificar aplicación

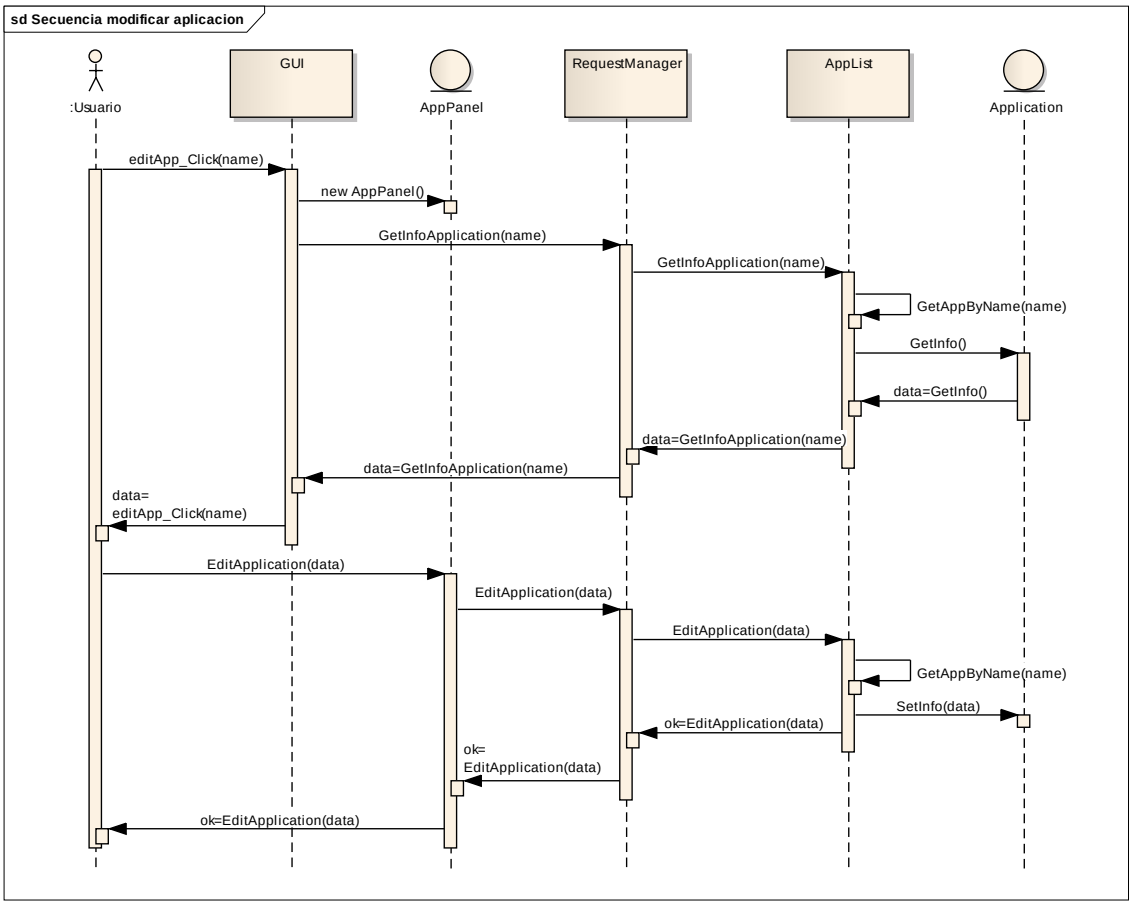

Figura 28: Secuencia modificar aplicación

La Figura 28 muestra un ejemplo del caso de uso en el que un usuario decide modificar una aplicación creada anteriormente en el sistema. En primer lugar, el usuario elegirá la opción de modificar una aplicación cuyo nombre es “name”. Esta acción se dividirá a su vez en dos acciones: por un lado la interfaz mostrará una nueva pantalla para representar la información, y por otro lado, el sistema devolverá la información almacenada en el sistema que está asociada a la misma. Para esto último, se realizará una llamada al método **GetInfoApplication(name)**. Como resultado final, el usuario obtendrá una nueva pantalla en la que aparecerán todos los parámetros asociados.

Una vez que el usuario haya modificado los valores oportunos de la aplicación y haya confirmado el proceso, se realizará una llamada al método **EditApplication(data)** que sobreescribirá la información anterior por los nuevos datos procedentes del usuario.

Borrar aplicación

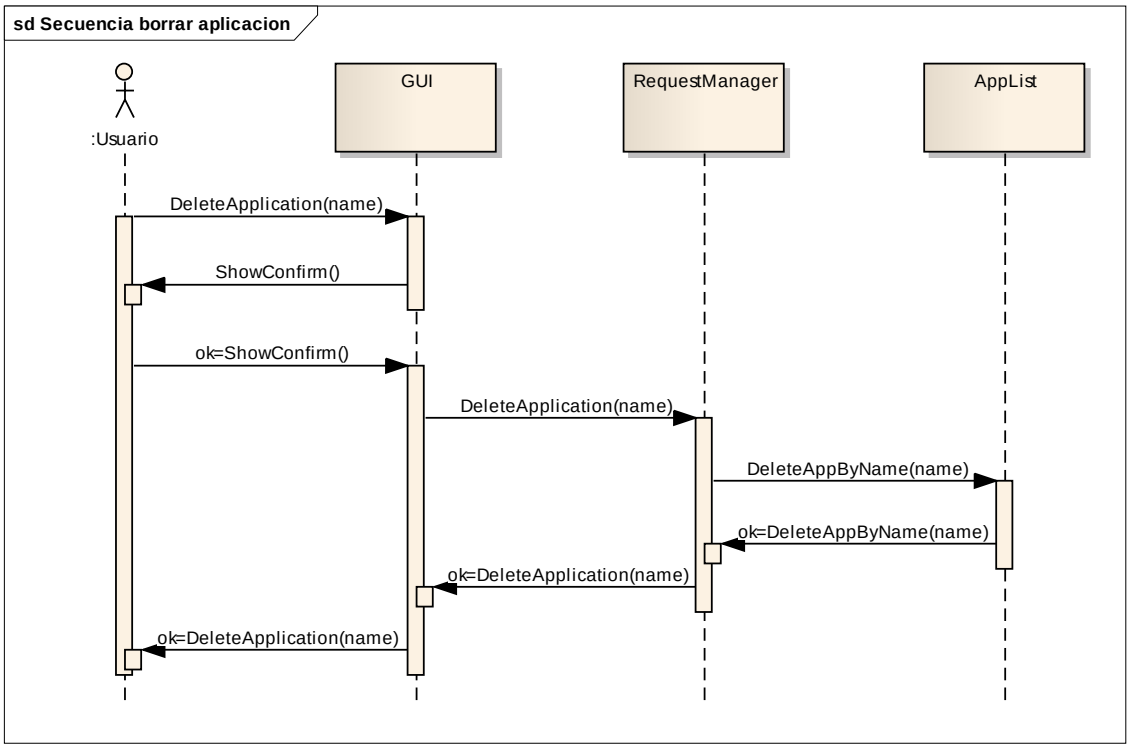

Figura 29: Secuencia borrar aplicación

La anterior Figura 29 representa la secuencia de acciones que se realizan en el caso de que el usuario decida eliminar una aplicación creada anteriormente en el sistema. Cuando el usuario le indica al sistema que desea borrar una determinada aplicación (para lo que se enviará el nombre de la misma), el sistema le pedirá una confirmación (como pasa con todos los procesos de borrado). En el caso de que el usuario acepte esta petición, se realizará una llamada al método **DeleteApplication(name)** del gestor de peticiones, por lo que se eliminará la misma y se le notificará el resultado al usuario.

En la Figura 30 aparece reflejado el caso en el que un usuario se dispone a eliminar todas las aplicaciones almacenadas en el sistema. Para ello, el usuario realiza una llamada al método **DeleteAllApps()** que provoca que el sistema le devuelva una petición de confirmación. Si el usuario confirma esta acción, se realiza la llamada al propio método **RemoveAllApps()** del gestor de peticiones (clase *RequestManager*) que se encargará de realizar otras llamadas que

harán que se elimine la información de todas las aplicaciones almacenadas en el sistema. Finalmente, se devolverá un mensaje informando de la situación al usuario.

Borrar todas las aplicaciones

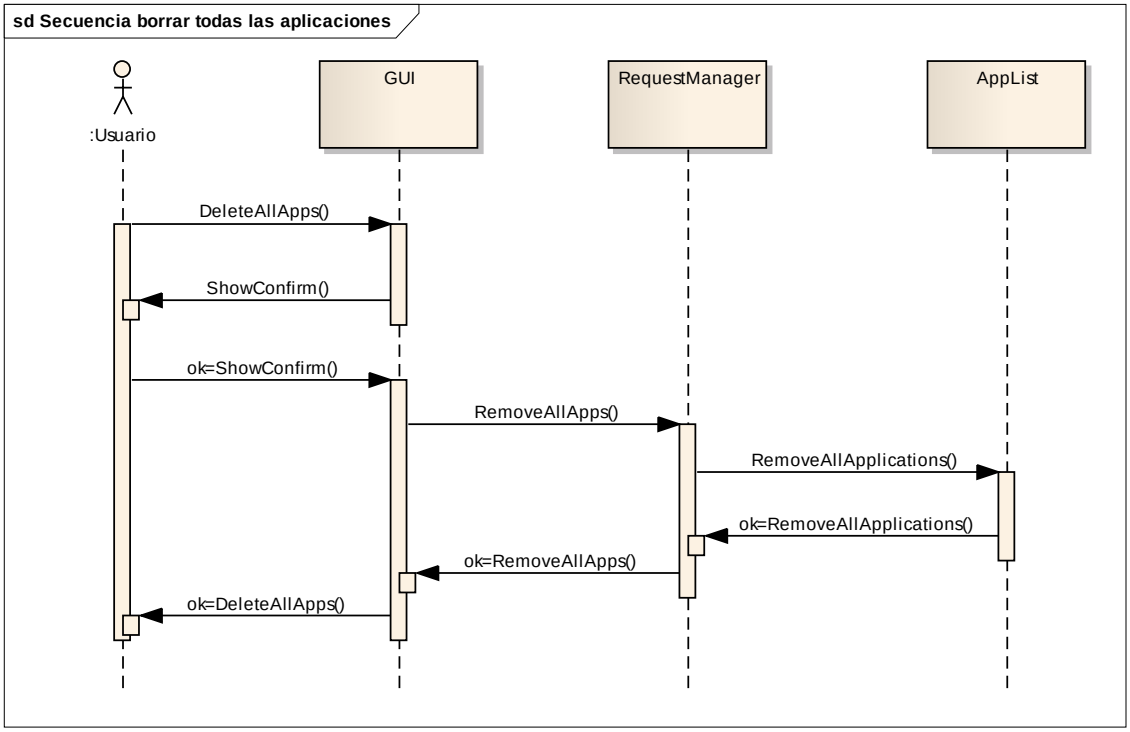

Figura 30: Secuencia borrar todas las aplicaciones

La Figura 31 representa un escenario típico de creación de un nuevo cloud por parte de un usuario del sistema. Cuando el usuario interactúa con la interfaz para generar un nuevo cloud se crean una serie de llamadas: por un lado, la interfaz genera una nueva pantalla que servirá para la inserción de datos, y por otro lado, esta pantalla debe mostrar al usuario toda la información que tiene disponible para la generación de este cloud.

Un cloud se compone de un componente hardware que está formado por una serie de maquinas virtuales y un componente software, formado por un conjunto de tareas o trabajos de usuario. Por ello, es necesario que en el momento de la creación del cloud, el usuario tenga constancia de todas las máquinas virtuales y todas las tareas creadas con anterioridad, para así poder vincularlas al nuevo cloud.

Así, la primera interacción del usuario con la interfaz para solicitar la creación de un nuevo cloud, genera tres llamadas a distintos métodos del gestor de peticiones del sistema, como son **GetMachineNames(), GetAppNames()** y **GetAppTypes()**. La primera devuelve los nombres de

las distintas máquinas virtuales creadas con anterioridad en el sistema; mientras que la segunda y la tercera devuelven los distintos nombres y tipos de todas las tareas almacenadas en el sistema.

Crear cloud

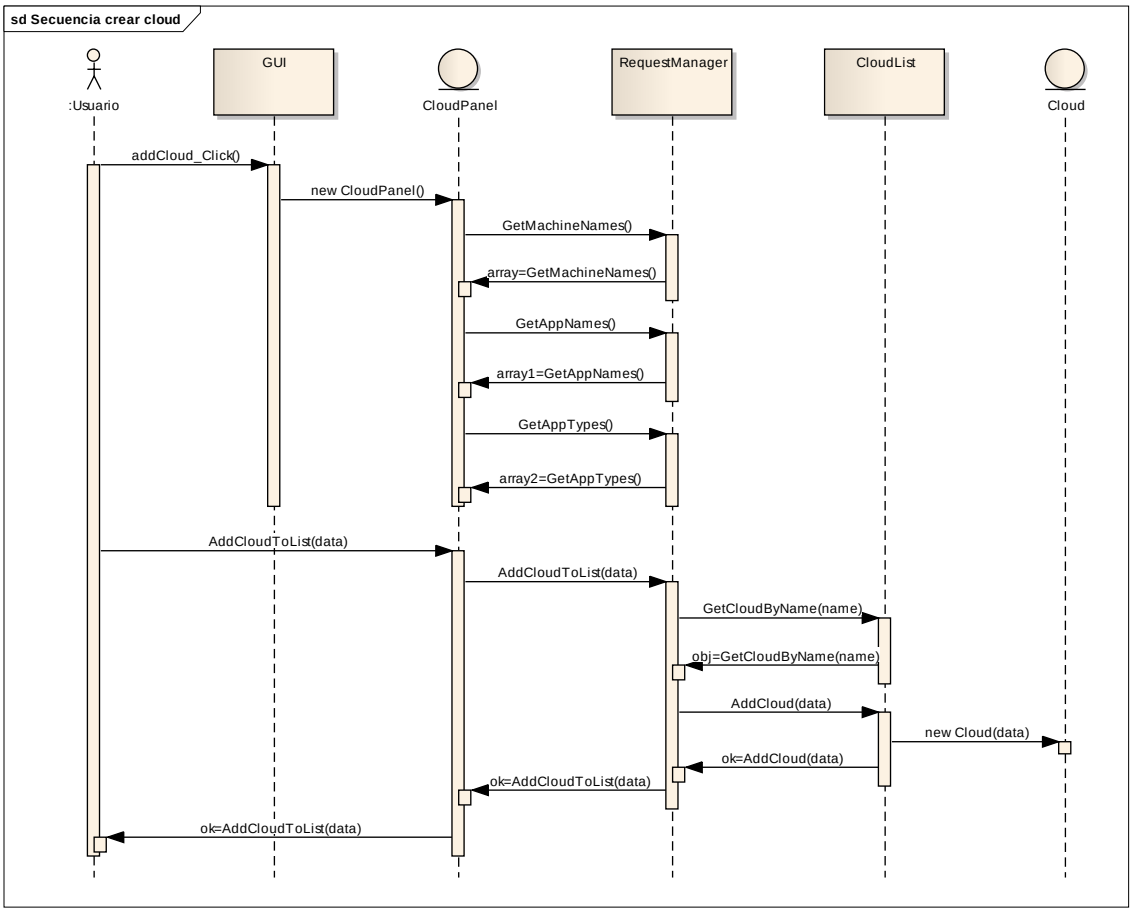

Figura 31: Secuencia crear cloud

Una vez que el usuario haya configurado adecuadamente su nuevo cloud y haya rellenado los campos necesarios podrá confirmar la creación del cloud. Esta acción provocará una llamada al método **AddCloudToList(data)** del gestor de peticiones, el cual deberá a su vez comprobar la consistencia del sistema antes de crearlo. Por ello, realizará una llamada al método **GetCloudByName(name)** de la clase *CloudList* y solo creará el nuevo objeto en el caso de que no haya ningún otro creado anteriormente con el mismo nombre. Finalmente, se mostrará un mensaje de notificación al usuario con el resultado del proceso.

Modificar cloud

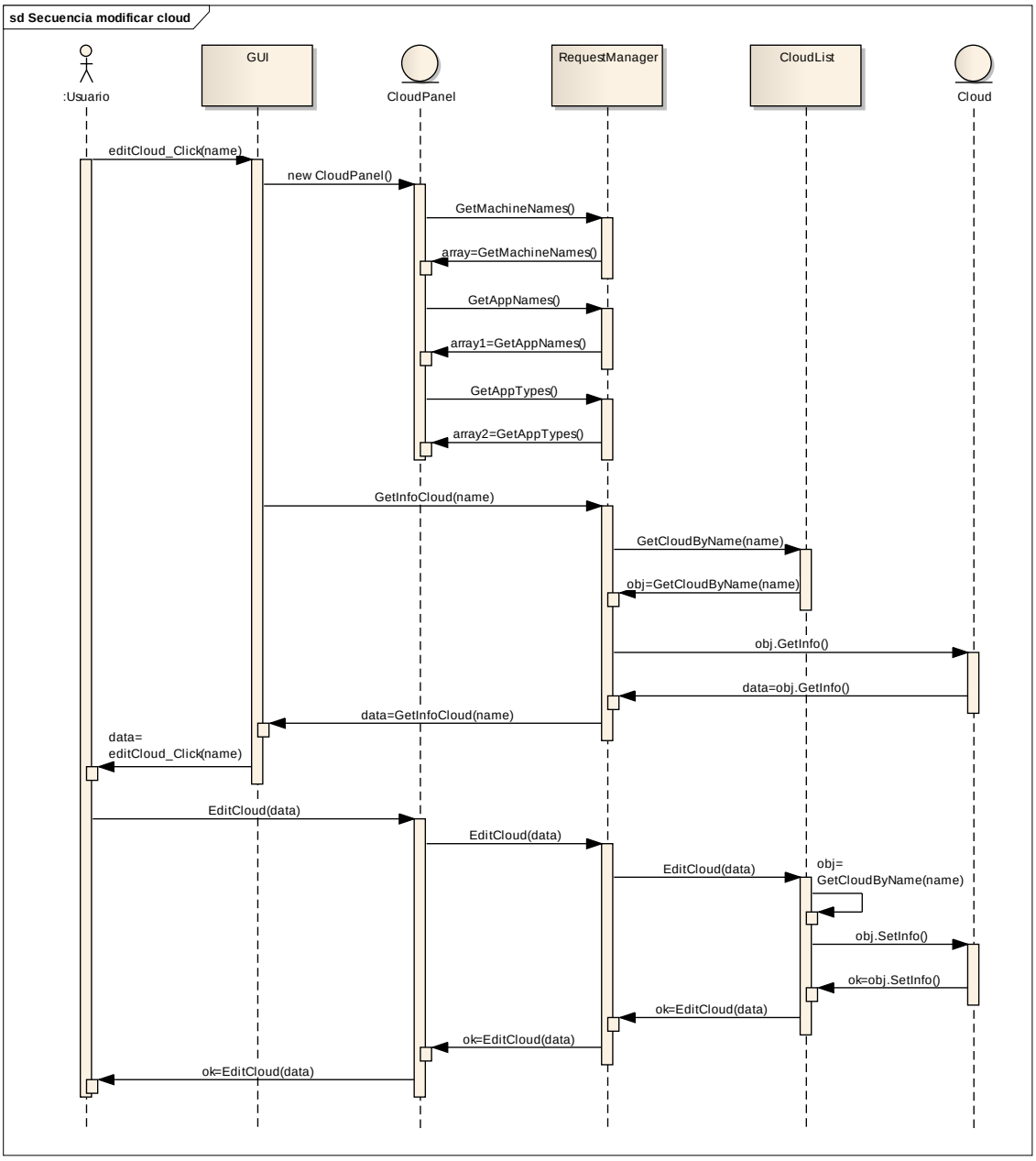

Figura 32: Secuencia modificar cloud

Por su parte, la Figura 32 representa un escenario típico del caso en el que un usuario decide modificar los elementos de un determinado cloud creado con anterioridad en el sistema. Inicialmente, el usuario realizará una petición al sistema para modificar un cloud concreto indicando de cuál se trata a partir de un identificador (su nombre). Por ejemplo, en la figura anterior se va a modificar un objeto con el identificador “name”. Esta petición por parte del usuario producirá que se lleven a cabo una serie de acciones como veíamos en el caso de la

creación anterior: se mostrará una pantalla de contenidos para representar los datos del cloud y se cargarán los distintos elementos que podemos utilizar en su gestión mediante las llamadas a los métodos **GetMachineNames(), GetAppNames()** y **GetAppTypes()**. Pero además de esto deberán cargarse los datos almacenados actualmente para el cloud en cuestión, por lo que se realiza otra llamada al método **GetInfoCloud()** del gestor de peticiones, el cual devuelve la información requerida. Como resultado de esta primera fase, el usuario tendrá en pantalla toda la información del elemento que quiere modificar.

Una vez que el usuario haya modificado la información deseada del cloud y haya aceptado los cambios, se llamará al método **EditCloud(data)** con la nueva información que se debe asociar. Como resultado de esta llamada se modificarán los parámetros deseados en el sistema y se devolverá al usuario un mensaje de notificación con el resultado del proceso.

Borrar cloud

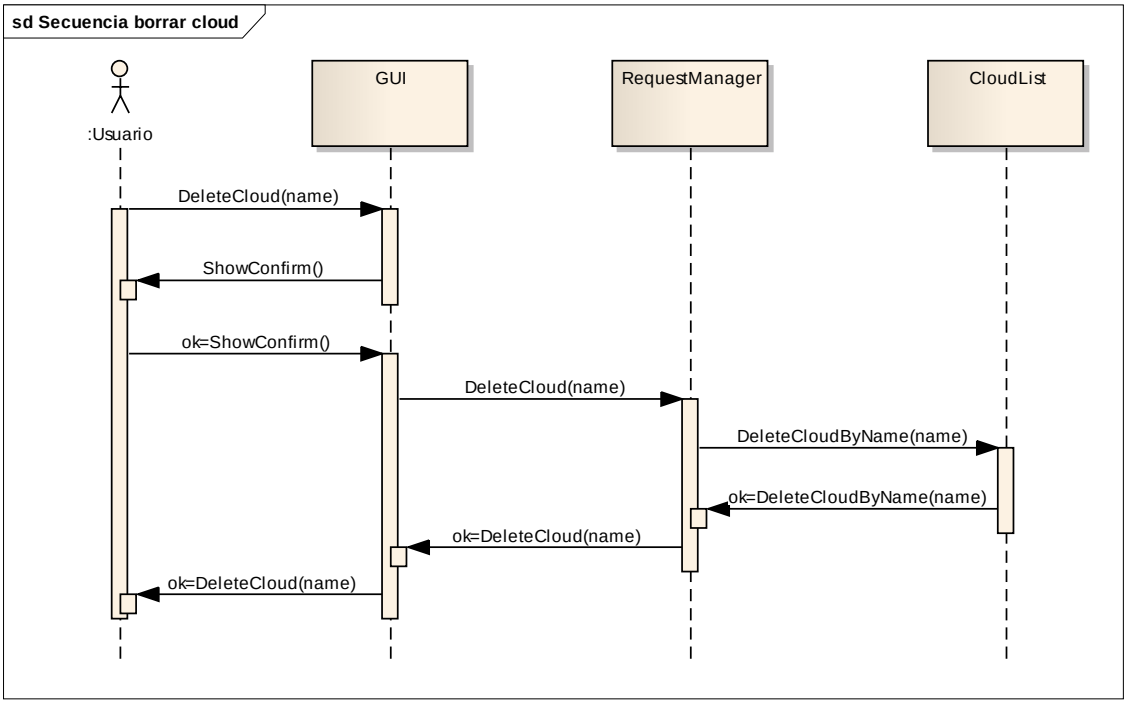

Figura 33: Secuencia borrar cloud

La Figura 33 simboliza la secuencia de acciones propia de un proceso de borrado de un determinado cloud dentro del sistema. Inicialmente el usuario selecciona el cloud que desea eliminar del sistema e interactúa con el sistema enviando dicha información. El sistema por su parte, responde a la acción enviando un mensaje de petición de confirmación al usuario. En el caso de que el usuario responda afirmativamente a esta petición, se llamará al método

DeleteCloud(name) del gestor de peticiones, y a partir de entonces se llevará a cabo la eliminación del mismo del sistema y se enviará al usuario un mensaje de notificación con el resultado del proceso.

Borrar todos los clouds

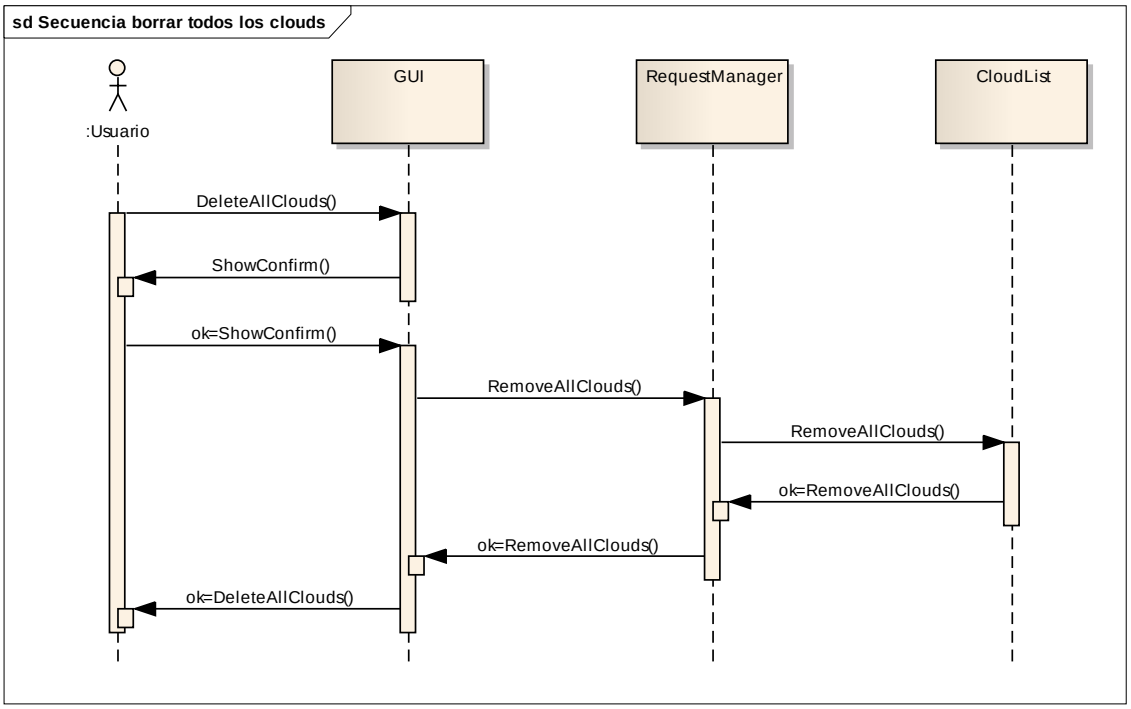

Figura 34: Secuencia borrar todos los clouds

La Figura 34 representa un escenario típico en el que un usuario del sistema se prepara para eliminar todos los clouds almacenados en el sistema. El usuario realizará una llamada al método **DeleteAllClouds(),** la cual provocará que el sistema le devuelva un mensaje de petición de confirmación del borrado. En el caso de que el usuario acepte finalmente este borrado, se realizarán una serie de llamadas que producirán la eliminación definitiva de todos los clouds almacenados en el sistema. Para finalizar, se enviará un mensaje al usuario informándole del resultado de todo el proceso.

En la Figura 35 se muestra un ejemplo de situación en la que el usuario decide crear una nueva tarea en el sistema. Para ello, elige la opción “addUser” de la interfaz (en la GUI, las tareas de usuario se han definido simplemente como “Users” o usuarios) lo que le devolverá un nuevo panel donde ingresar los datos oportunos. Después, el usuario deberá confirmar la creación, lo cual provocara una llamada al método **AddUserToList(data)** de la clase RequestManager. A continuación, el sistema comprobara que la tarea de usuario no ha sido creada con

anterioridad mediante una llamada al método **GetUserByName(name)**. En el caso de que no haya sido creado, se realiza otra llamada al método **AddUser(data)** lo que conlleva la creación de un nuevo objeto de la entidad User, el cual se añade a la lista de tareas de usuario creadas.

Crear tarea de usuario

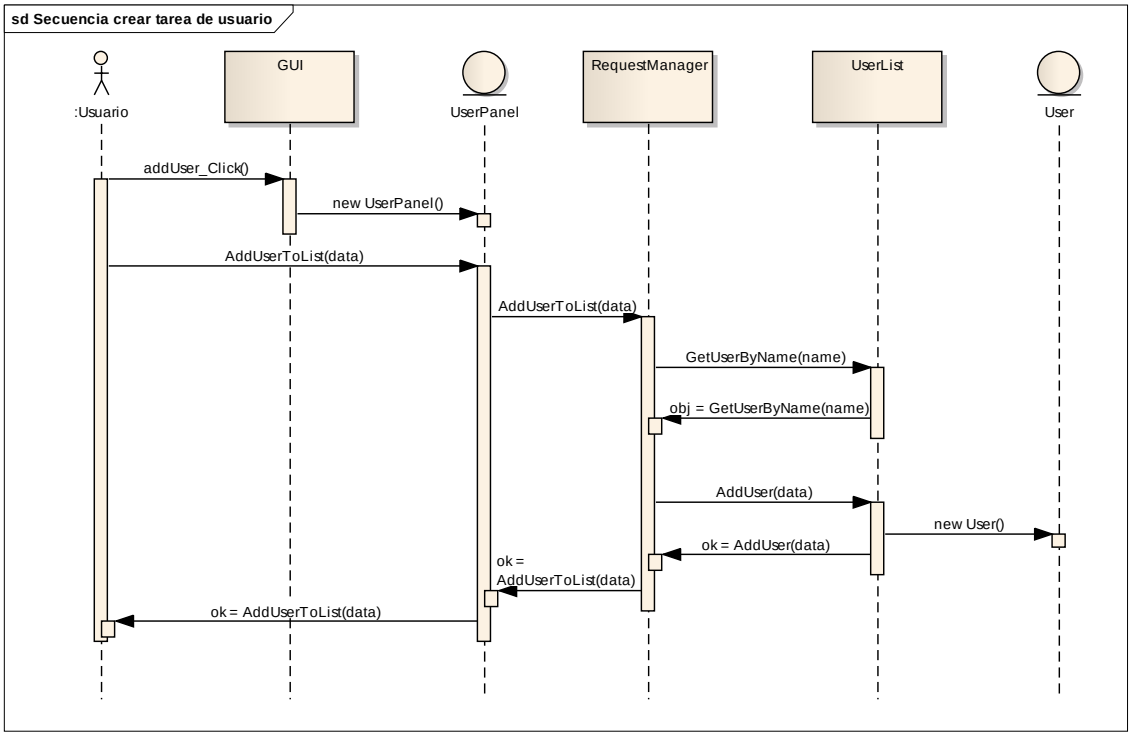

Figura 35: Secuencia crear tarea de usuario

A continuación aparece una ilustración que representa el proceso por el cual se modifica en el sistema una tarea anteriormente creada por un usuario. En la Figura 36 se puede apreciar como la acción se genera cuando un cliente del sistema activa el evento "editUser_Click(name)" sobre el nombre de la tarea de usuario que se pretende modificar. Esto provoca que se abre un nuevo panel y que se realice una llamada al método **GetInfoUser(name).** Mediante este procedimiento se busca en la lista de tareas de usuario generadas (clase *UserList*) aquella que tiene como identificador "name" (mediante **GetUserByName(name)**) y se obtienen su información, mostrándola en el nuevo panel.

Una vez que el cliente ha modificado los datos deseados y ha confirmado el proceso, se realiza una llamada al método **EditUser(data)**, el cual obtiene de nuevo el trabajo de usuario de la lista anterior y establece los nuevos parámetros mediante una nueva llamada al método **SetInfo(data)**.

Finalmente, se devuelve una confirmación del proceso al usuario.

Modificar tarea de usuario

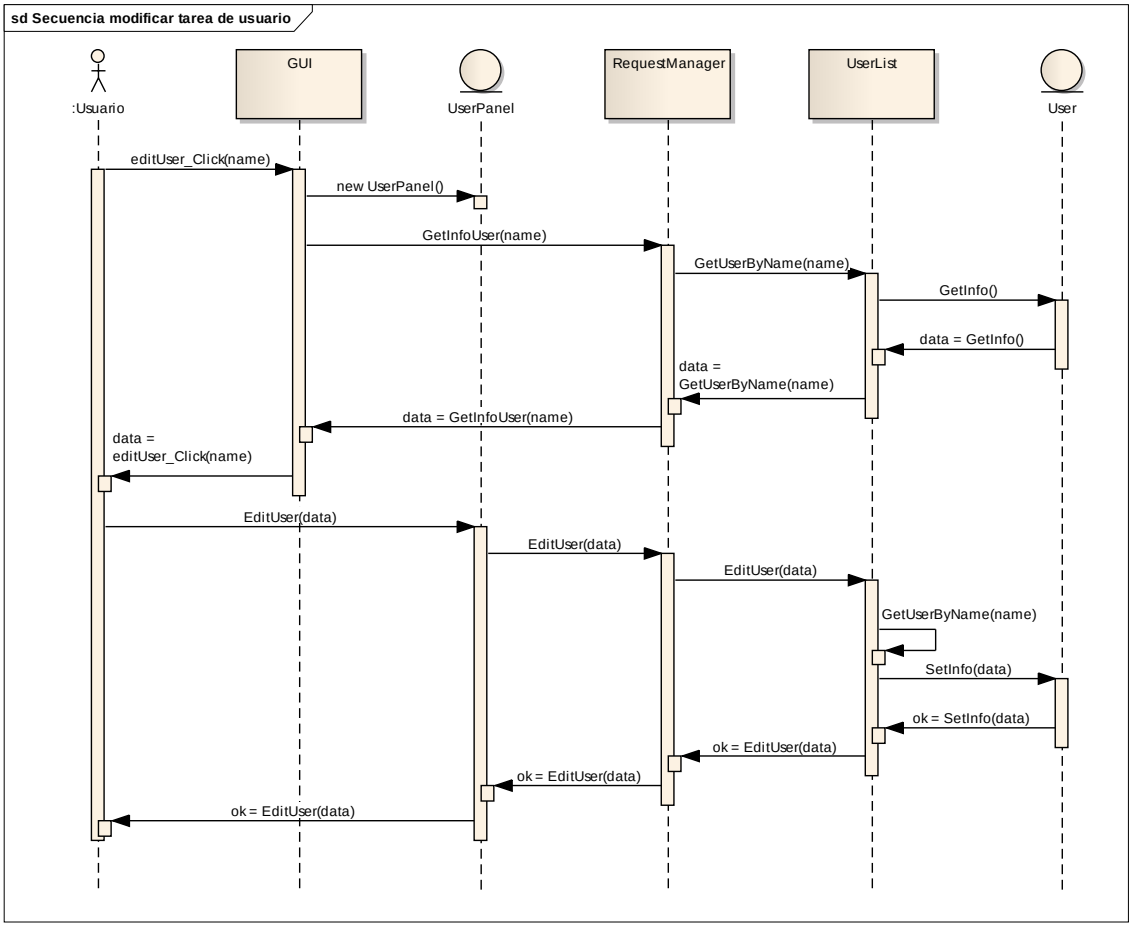

Figura 36: Secuencia modificar tarea de usuario

La Figura 37 representa un escenario típico en el que un usuario desea borrar una tarea creada previamente en el sistema. Para ello el usuario debe seleccionar la tarea escogida y activar el evento que provoca la llamada al método **DeleteUser(name)**. Después de esto, el sistema le enviara un mensaje solicitando confirmación para el proceso de borrado de la tarea en el sistema.

Si el usuario confirma el borrado, se realizara una llamada al método **DeleteUserByName(name)** de la clase UserList (la lista de tareas de usuario). Tras esto, se devolverá un mensaje de confirmación al cliente informando sobre el éxito de la operación.

Borrar tarea de usuario

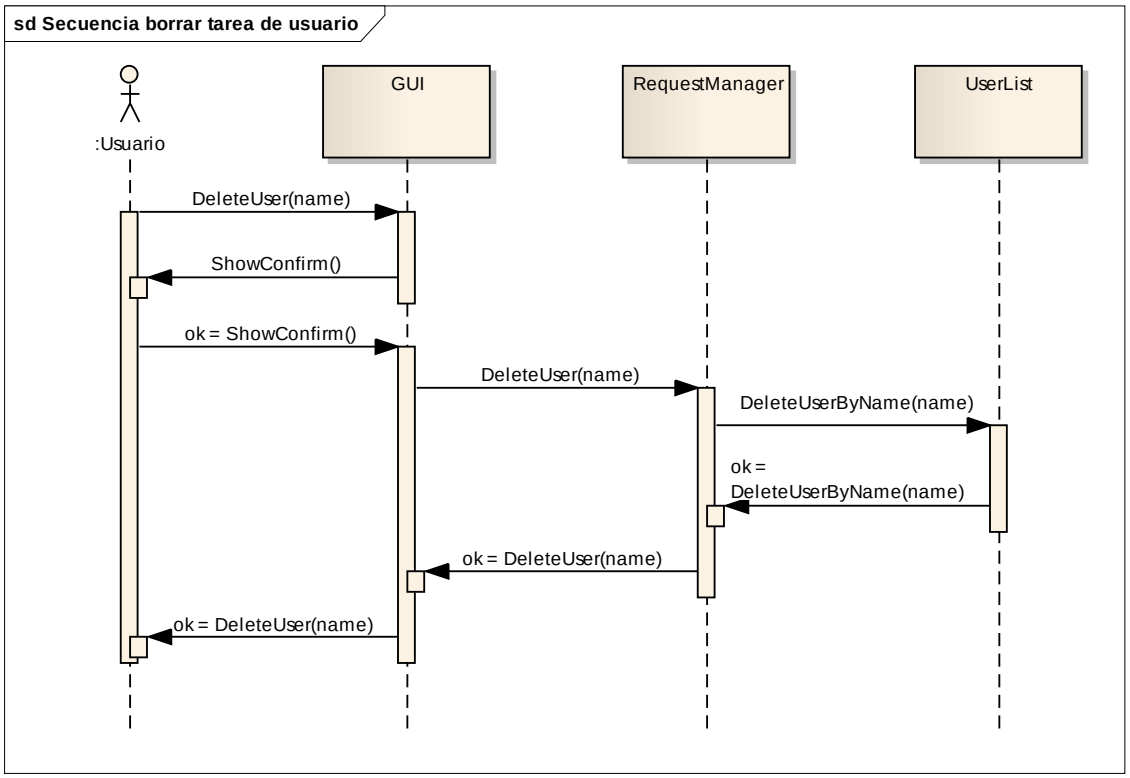

Figura 37: Secuencia borrar tarea de usuario

En la ilustración que aparece a continuación se muestra un caso típico en el que un usuario de la aplicación desea borrar todas las tareas o trabajos que han sido anteriormente generados en el sistema. La Figura 38 muestra como, cuando el usuario activa el evento necesario para borrar todas las tareas, la GUI (o interfaz grafica de usuario) le devuelve un mensaje de petición de confirmación.

En el caso de que el usuario confirme el borrado de todas las tareas, se realiza una llamada al método **RemoveAllUsers()** del gestor de peticiones (clase *RequestManager*). Este, a su vez, realizara otra llamada al método del mismo nombre de la clase UserList, que se encargara de borrar la lista de tareas de usuario del sistema.

Borrar todas las tareas de usuario

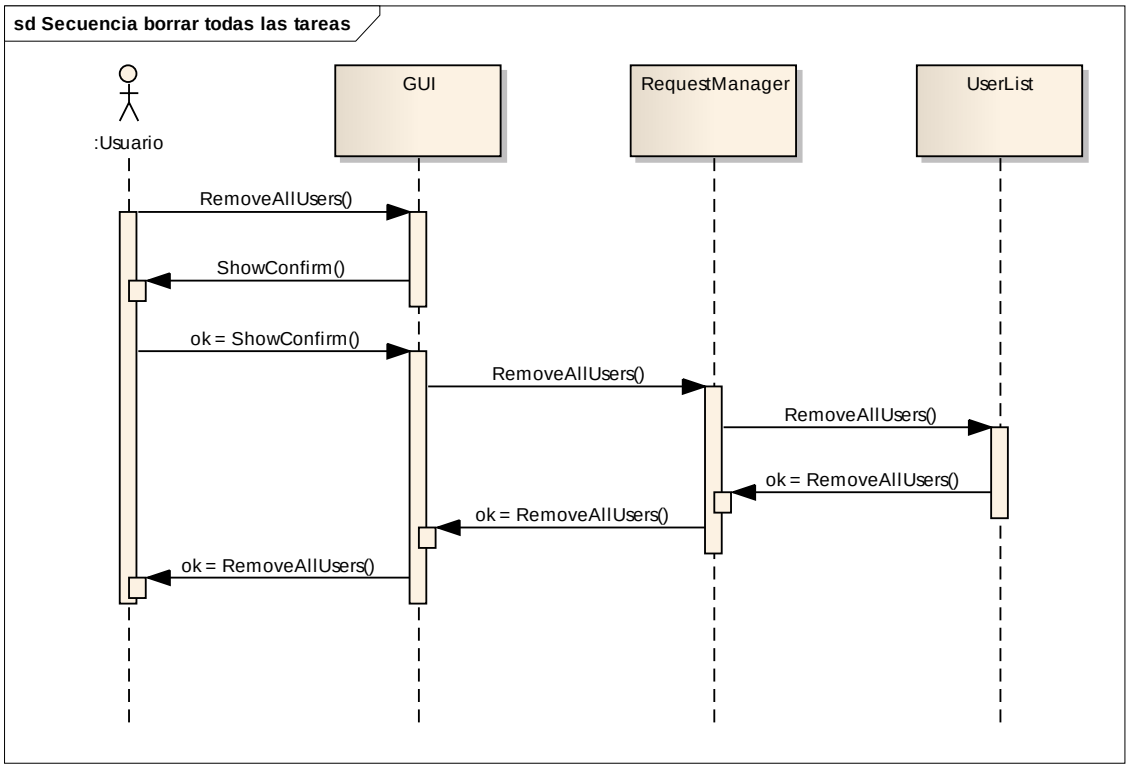

Figura 38: Secuencia borrar todas las tareas de usuario

Por su parte, la Figura 39 que aparece a continuación muestra un ejemplo de situación en la que un usuario de la aplicación lanza una simulación de un cloud y se generan los ficheros de configuración necesarios para dicha prueba. Para lanzar una simulación el usuario debe seleccionar un cloud debidamente parametrizado. Una vez que se escoge la opción de simular el cloud se realiza una llamada al método **CreateConfigFile(name)** del gestor de peticiones. Este método se encargará de dos tareas: por un lado, obtener la información asociada al cloud en cuestión (ya que el usuario solo envía una referencia al nombre) por lo que se realiza una llamada al método **GetCloudByName()**, y por otro lado, se realiza una llamada al método **CreateConfigFiles(obj)** con la información del cloud que hemos obtenido en el paso anterior. Este método de la clase *FileSystem* se encarga de traducir la información del cloud en código inteligible para el simulador (mediante el método **GenerateConfigFile(obj)**) y de generar una serie de ficheros de configuración necesarios para que la realización de la simulación sea satisfactoria.

Generar ficheros de configuración

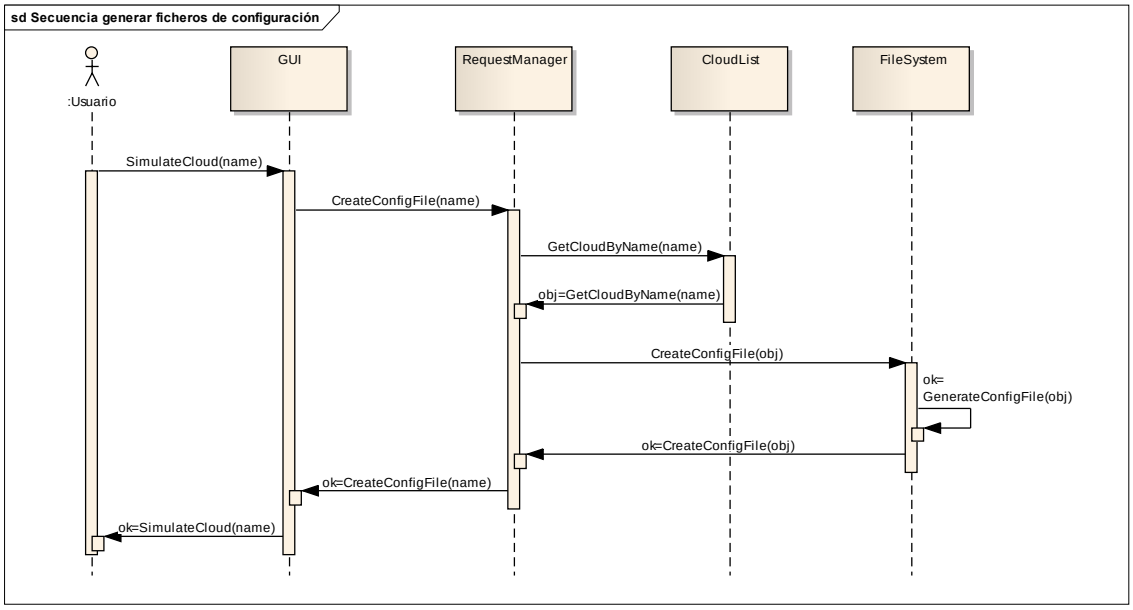

Figura 39: Secuencia generar ficheros de configuración

Lanzar simulación

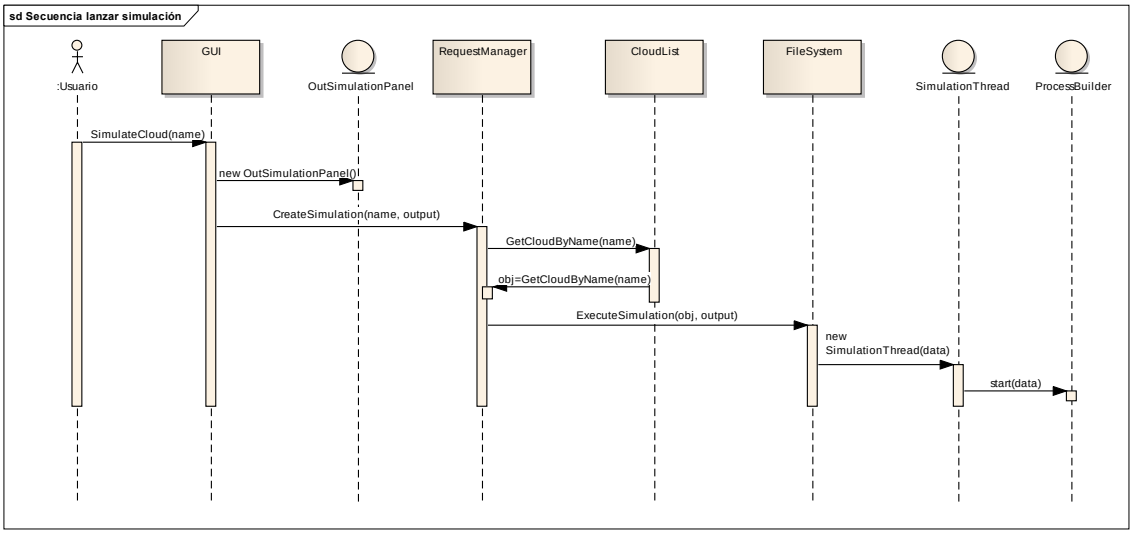

Figura 40: Secuencia lanzar simulación

En la Figura 40 se representa un escenario típico en el que un usuario de la aplicación lanza una simulación de un cloud concreto. Éste es un escenario consecutivo al anterior en el que se generaban los ficheros de configuración de la propia simulación. De esta forma, cuando un usuario elige la opción de simular un cloud debidamente configurado y se realiza la llamada al método **SimulateCloud(name)**, tras la cual se suceden una serie de acciones, que aparecen representadas en la figura anterior.

Así, primero se crea una pantalla en la interfaz donde el usuario podrá ir siguiendo la ejecución paso a paso y posteriormente se llama al método encargado de lanzar la simulación en sí, llamado **CreateSimulation(name),** pasándole el panel que acabamos de crear. De esta forma, se define la salida de la traza que devolverá el simulador y el usuario tendrá información en tiempo real de cómo esta discurriendo el proceso.

El método anterior tiene dos funciones principales en su ejecución: por un lado, se buscan los datos del cloud que queremos simular mediante el método **GetCloudByName(name)** y por otro lado se envía esa información mediante otra llamada al método **ExecuteSimulation(obj, output)** de la clase *FileSystem*. Esta clase se encargará de crear un nuevo hilo (instancia de *SimulationThread*) que será el que realmente haga la llamada al simulador con los datos oportunos.

La Figura 41 muestra un ejemplo de secuencia en la que el sistema monitoriza la información en tiempo real de la simulación. De esta forma, la aplicación se comunica directamente con el usuario (a través del componente *OutSimulationPanel*) para mostrarle datos de ejecución.

Cuando se realiza la llamada al nuevo hilo para que lance la simulación (método **start()**) se le asigna un listener a su salida, por lo que se va leyendo línea a línea y se devuelve dicho valor a la instancia OutSimulationPanel a la que tiene acceso el usuario desde la interfaz. Así se muestra el estado de la simulación en tiempo real.

Monitorizar simulación

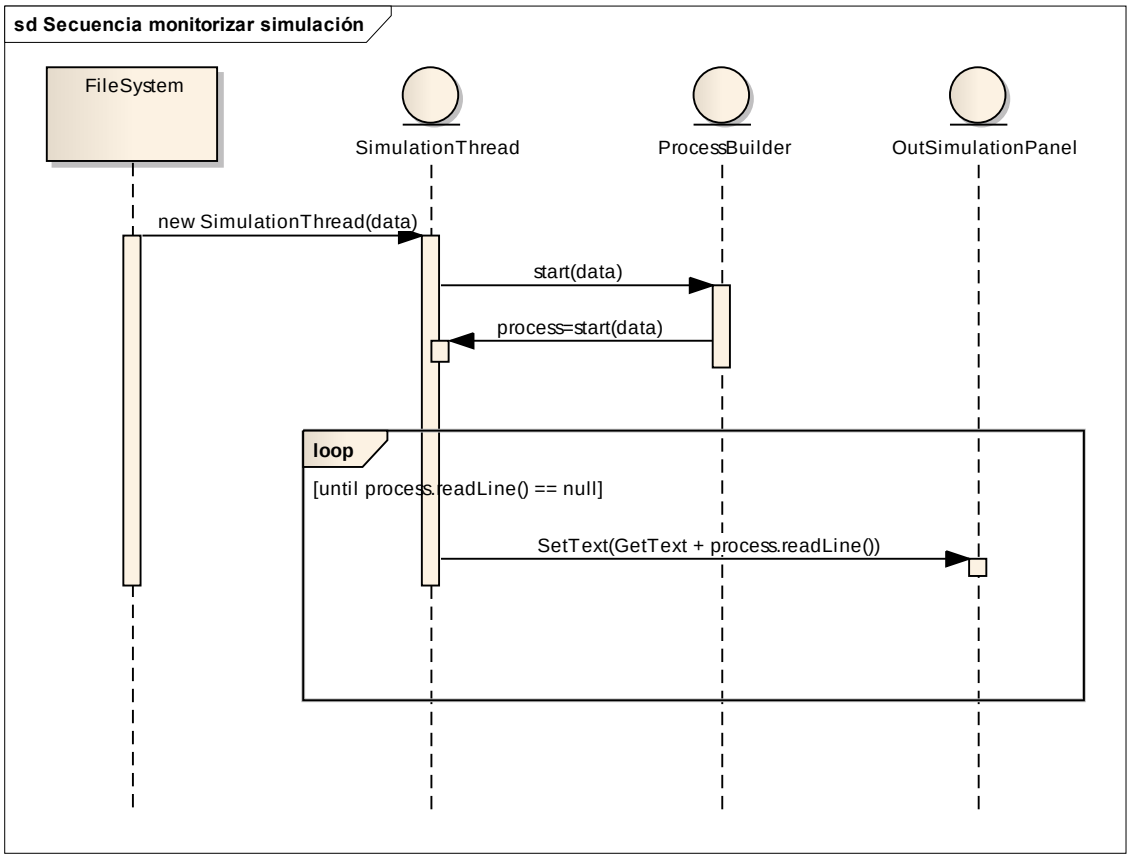

Figura 41: Secuencia monitorizar simulación

El ejemplo de la Figura 42 muestra una secuencia en el que el usuario de la aplicación le indica a la misma que lea los resultados de la simulación para preparar la generación de informes.

Cuando el usuario elige la opción de ver los resultados de la simulación en envía en primer lugar un evento que finalmente provoca que el gestor de peticiones haga una llamada al método **ReadCloudResults(name)** del componente *FileSystem*, el cual devuelve un array con toda la información leída de fichero. Posteriormente, se envía esta información al componente *ResultManager* para que esté disponible para la generación de gráficas e informes.

Interpretar resultados simulación

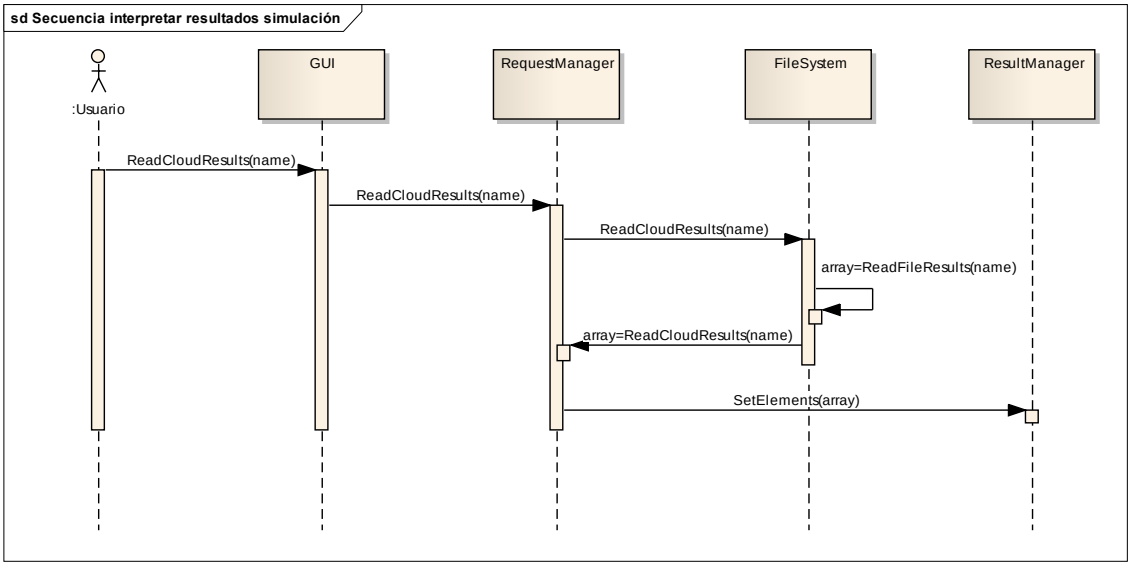

Figura 42: Secuencia interpretar resultados simulación

Por su parte, en la Figura 43 que aparece a continuación, se muestra un ejemplo de escenario en el que el usuario decide visualizar los resultados obtenidos de una determinada simulación.

Después de que se haya reproducido el escenario anterior, tras la elección de la opción de visualización de los resultados por parte del usuario, se abre una nueva pantalla que dará cabida a estos resultados y se realizará una llamada al método **GenerateGraph(name)** con el nombre del cloud simulado. El hilo de ejecución llegará hasta el componente *ResultManager*, en el que antes habíamos guardado los resultados de la simulación, el cual irá leyendo dichos resultados y generando una serie de gráficos de acuerdo a la cantidad de información leída, guardando los mismos en distintos ficheros físicos. Después de esto, la ejecución volverá a la interfaz de usuario que cargará estos gráficos generados para su visualización.

Generar gráficas

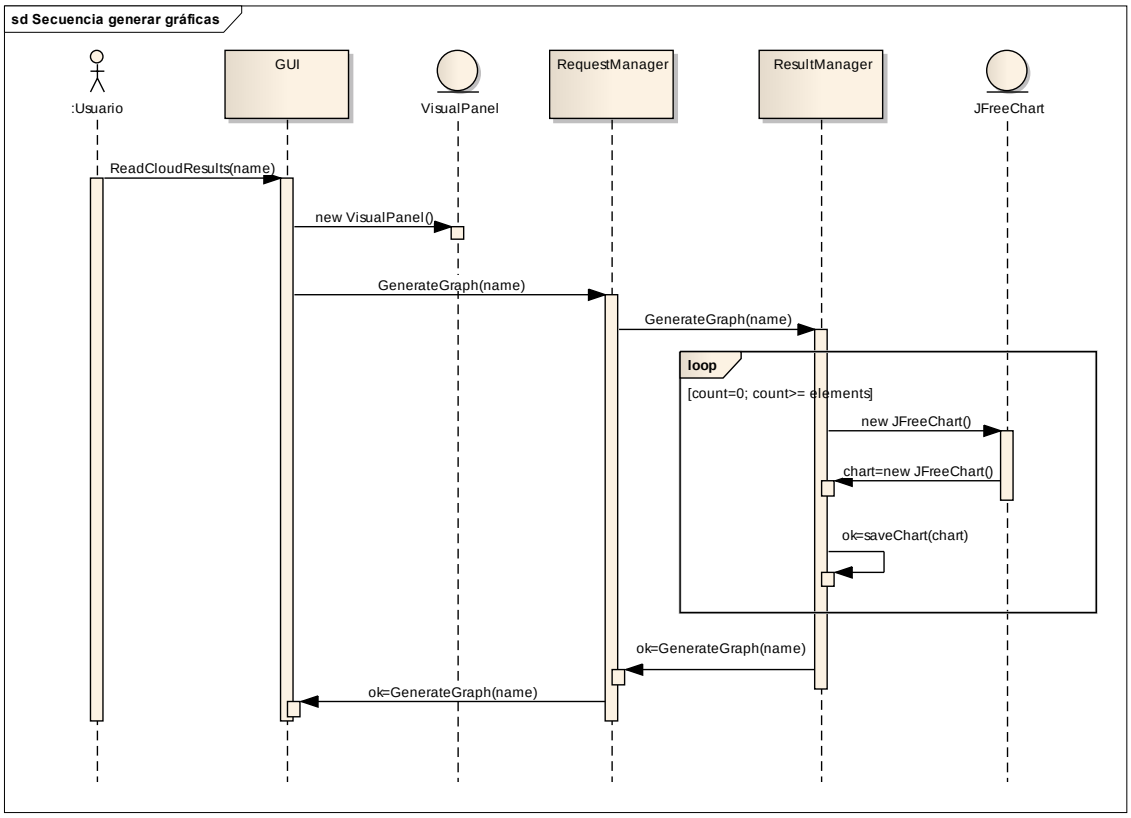

Figura 43: Secuencia generar gráficas

La Figura 44 representa el escenario en el que un usuario desea generar el informe asociado a una simulación ya finalizada. La aplicación debe ser capaz de generar un informe por cada cloud que haya sido simulado por el usuario. Así, una vez que han sido interpretados los resultados de la simulación y se han generado las distintas gráficas asociadas a dichos resultados, el usuario puede elegir la opción de generar un informe detallado de la simulación realizada.

Cuando se seleccione esta opción, la aplicación devolverá un cuadro de diálogo donde el usuario deberá seleccionar el nombre del fichero PDF a crear y la ubicación del mismo. En el momento en el que el usuario complete estos datos, se realizará la llamada al método **GeneratePDF(name, path)** del gestor de peticiones. Este método, buscará los datos asociados al cloud del que se quiere generar el informe mediante la llamada **GetCloudByName(name)** y una vez tenga esos datos se realizará una llamada al método **GeneratePDF(obj, path)** de la clase *FileSystem* (encargada de todo el trabajo de entrada/salida de ficheros). Este último método será el que genere el fichero PDF y le dé el formato apropiado.

Generar informes

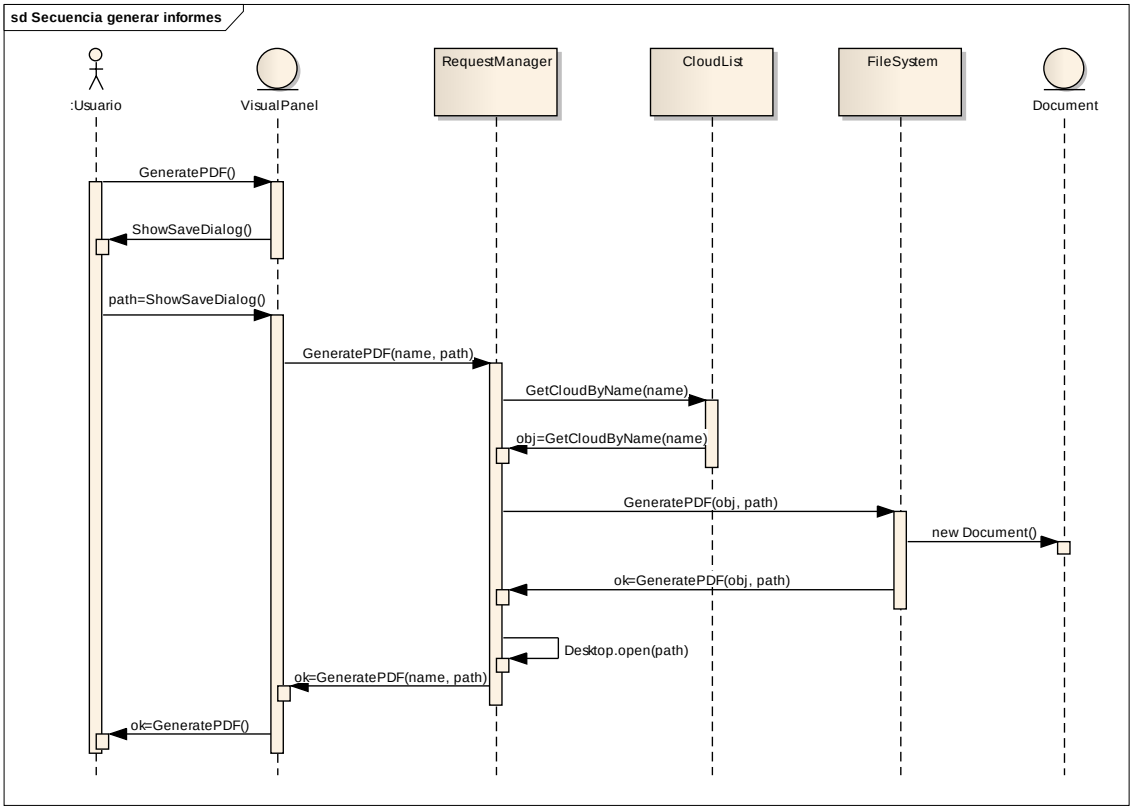

Figura 44: Secuencia generar informes

Tras la creación del fichero en la ruta determinada, será el propio gestor de peticiones el encargado de acceder al servicio oportuno del SO (sistema operativo) para abrir dicho fichero y mostrarlo al usuario.

La Figura 45 muestra el escenario típico de actuación donde el usuario se dispone a guardar su configuración personal en el sistema. Este almacenamiento de información incluye todos los datos referentes a los elementos creados por el usuario en la última sesión de trabajo junto con los que hubiera almacenado de sesiones anteriores. Estos elementos se dividen, como hemos visto anteriormente, en máquinas virtuales, tareas, aplicaciones y clouds generados.

Guardar configuración

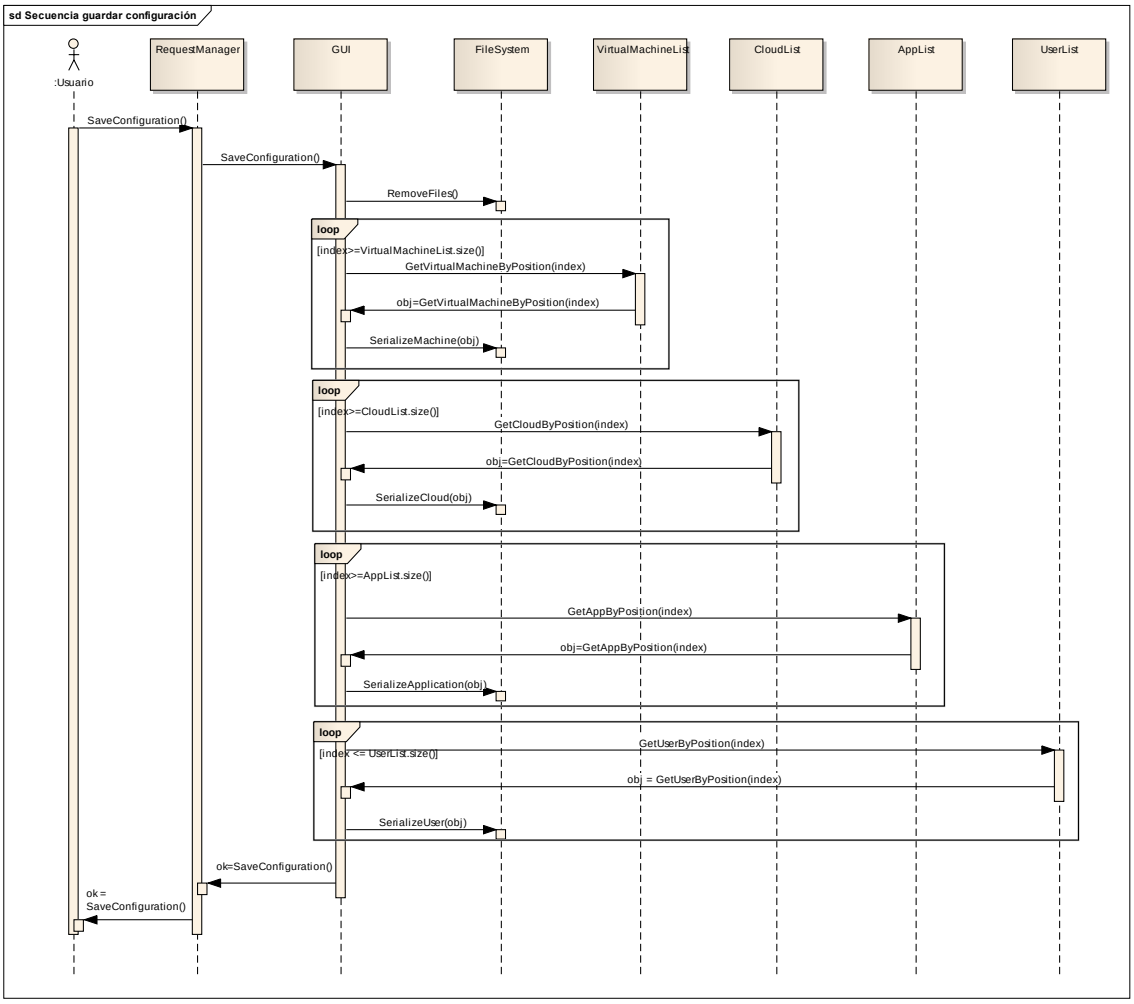

Figura 45: Secuencia guardar configuración

Por ello, cuando el usuario elige la opción de guardar su configuración se realizan distintas llamadas a métodos del gestor de ficheros (clase *FileSystem*): por un lado, se borran todos los ficheros que hubiera guardados con anterioridad en la ruta por defecto mediante una llamada al método **RemoveFiles()**, y después se guarda toda la información de la aplicación, garantizando así la consistencia del sistema.

Para guardar todas las máquinas virtuales, todas las tareas, aplicaciones y clouds del sistema se actúa siempre del mismo modo: se recorre con un bucle todos los elementos de un determinado tipo, se obtiene el elemento y se serializa en un fichero. Por ejemplo, en el caso de las máquinas virtuales se va accediendo a cada instancia realizando una llamada al método **GetVirtualMachineByPosition(index)**, y una vez que se tiene el objeto en cuestión se serializa realizando una llamada al método **SerializeMachine(obj)**.

Cargar configuración

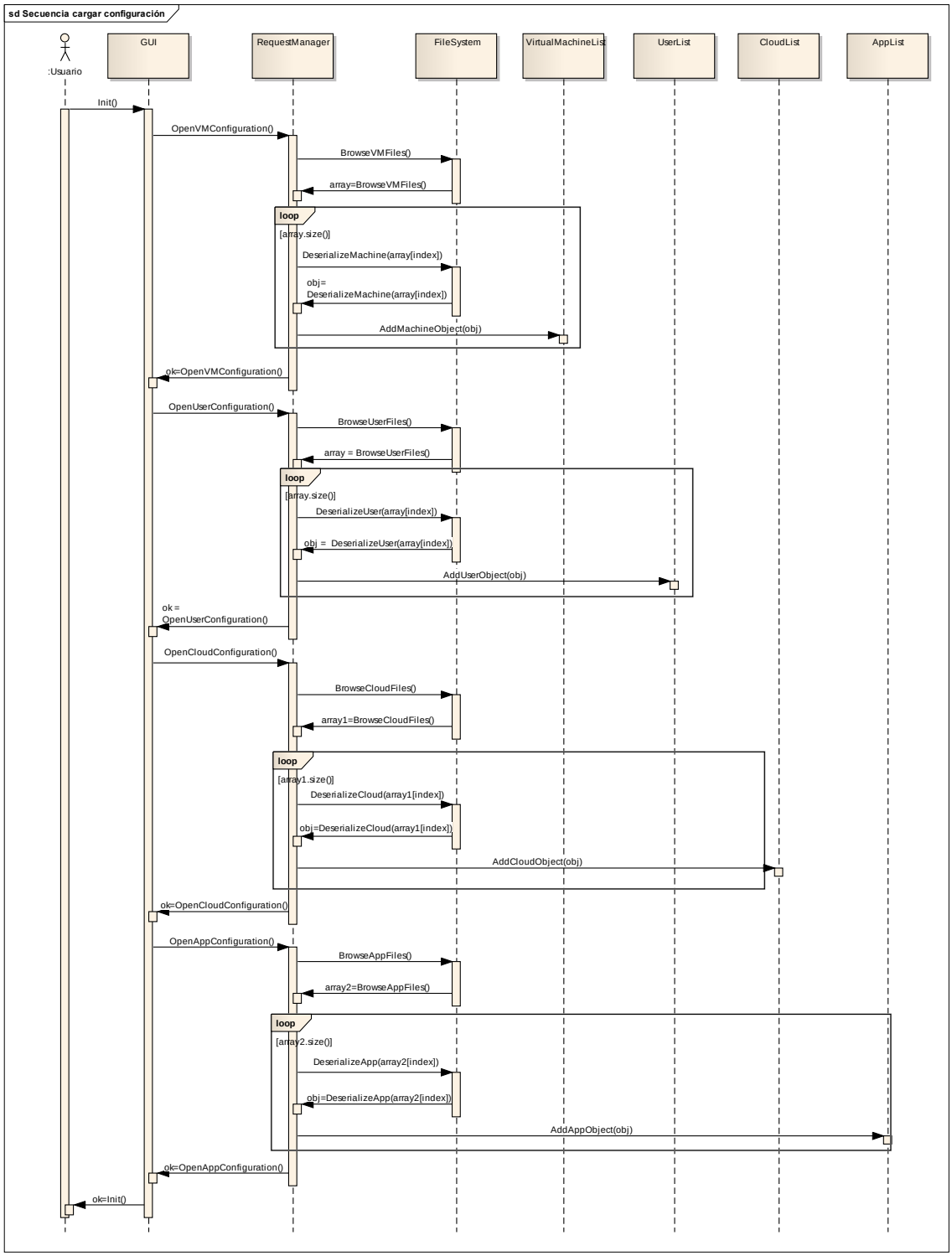

Figura 46: Secuencia cargar configuración

Después de actuar de manera similar para el resto de elementos del sistema, se devuelve un mensaje de notificación al usuario con el resultado del proceso.

Por su parte, la Figura 46 anterior muestra la secuencia de llamadas característica del caso en el que el usuario ejecute la aplicación y se cargue su configuración personal. Es importante señalar que esta acción se realiza de manera transparente al usuario, es decir, éste no interactúa con el sistema para decirle que cargue su configuración personal sino que esta carga se realiza automáticamente cada vez que ejecuta la aplicación.

Así pues, marcamos el punto de entrada de la secuencia como el método **Init()** que se ejecuta directamente cada vez que el usuario abre la aplicación. Este método, además de configurar e inicializar todos los elementos de la aplicación, se encarga de cargar la configuración de usuario en base al tipo de elemento almacenado. Primero se cargarán todas las máquinas virtuales almacenadas. Cada máquina virtual almacenada está representada por un fichero en disco, y todas las máquinas se encuentran en el mismo directorio. De esta manera, primero se leerán todos los ficheros del directorio en cuestión y después se deserializará cada fichero y se añadirá cada elemento a la lista pertinente.

Por ejemplo, en el caso de las máquinas virtuales, se realizará una llamada al método **BrowseVMFiles()**, el cual devolverá un conjunto de rutas a los ficheros de las máquinas virtuales que hay almacenadas en su directorio. Con esta información, se deserializa cada fichero llamando al método **DeserializeMachine(path)** y el objeto resultante se almacena en la lista correspondiente llamando al método **AddMachineObject(obj)**. En el caso de las máquinas virtuales, además de añadir el objeto a la lista de máquinas, se crea un nuevo tipo de máquina haciendo **new MachineType(name)**, aunque por simplicidad no se muestra en el escenario anterior.

Estas son las acciones derivadas de la llamada al método **OpenVMConfiguration()** del gestor de peticiones. Cuando finaliza la ejecución de este método se realizan otras llamadas a los métodos **OpenCloudConfiguration(), OpenUserConfiguration()** y **OpenAppConfiguration()** para cargar el resto de elementos en el sistema. Finalmente, se envía un mensaje de notificación al usuario.

4.4 Descripción de los componentes de diseño

Una vez que se ha definido el modelo lógico en el que aparecen todos los componentes del diseño, se hace necesario describir cada uno de estos componentes, detallando conceptos importantes como las interfaces que proporcionan a otros componentes, las relaciones de jerarquía existentes, su función dentro del sistema, etc. Se ha decidido agrupar los componentes por capas para facilitar así su comprensión al lector.

Los componentes que forman la capa de la vista son los siguientes:

IDENTIFICADOR	CV-001		
NOMBRE	**GUI**	TIPO	Clase
PROPÓSITO	El propósito de este componente es agrupar todas las funcionalidades que están directamente relacionadas con la interfaz de usuario.		
FUNCIÓN	La función de este componente es permitir al usuario realizar todas las acciones reflejadas en los casos de uso del sistema, además de monitorizar el estado del sistema.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.		
INTERFACES	Ofrece una interfaz a cada uno de los componentes de la vista (de CV-002 a CV-007) para que estos puedan enviarle datos del usuario.		

Tabla 32: Componente Vista 001

	CV-002		
NOMBRE	**MachinePanel**	TIPO	Clase
PROPÓSITO	Este componente tiene como propósito principal agrupar toda la funcionalidad que tiene que ver con la gestión de los parámetros propios de una máquina virtual.		
FUNCIÓN	La función de este componente consiste en permitir al usuario la edición y creación de nuevas máquinas virtuales y la gestión de los parámetros de configuración que forman estas estructuras.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	Este componente depende de la clase JPanel incluida en el paquete javax.swing de Java.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CV-001(GUI), habilitada para el envío de información del sistema.		

Tabla 33: Componente Vista 002

IDENTIFICADOR	CV-003		
NOMBRE	**CloudPanel**	TIPO	Clase
PROPÓSITO	Este componente tiene como propósito principal agrupar toda la funcionalidad vista en apartados anteriores relativa a la gestión de clouds dentro del sistema.		
FUNCIÓN	La función de este componente consiste en editar y dar de alta nuevos clouds dentro del sistema, así como permitir al usuario configurar todos los componentes que los forman.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	Este componente depende de la clase JPanel incluida en el paquete javax.swing de Java.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CV-001(GUI), habilitada para el envío de información del sistema.		

Tabla 34: Componente Vista 003

IDENTIFICADOR	CV-004		
NOMBRE	**AppPanel**	TIPO	Clase
PROPÓSITO	Este componente tiene como propósito principal agrupar toda la funcionalidad vista en apartados anteriores relativa a la gestión de aplicaciones dentro del sistema.		
FUNCIÓN	La función de este componente consiste en editar y dar de alta nuevas aplicaciones dentro del sistema, así como ofrecer al usuario la posibilidad de configurar los parámetros que conforman estas aplicaciones o elegir el tipo de aplicación asociada a una tarea.		
SUBORDINADOS	Este componente no tiene subordinados.		

DEPENDENCIAS	Este componente depende de la clase JPanel incluida en el paquete javax.swing de Java.
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CV-001(GUI), habilitada para el envío de información del sistema.

Tabla 35: Componente Vista 004

IDENTIFICADOR	**CV-005**		
NOMBRE	**UserPanel**	**TIPO**	Clase
PROPÓSITO	Este componente tiene como propósito principal agrupar toda la funcionalidad vista en apartados anteriores relativa a la gestión de tareas o trabajos de usuario dentro del sistema.		
FUNCIÓN	La función de este componente consiste en permitir la modificación y el proceso de alta de nuevas tareas dentro del sistema. De esta forma, podrá decidir las aplicaciones que desea que se ejecuten sobre un numero determinado de maquinas virtuales.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	Este componente depende de la clase JPanel incluida en el paquete javax.swing de Java.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CV-001(GUI), habilitada para el envío de información del sistema.		

Tabla 36: Componente Vista 005

IDENTIFICADOR	CV-006		
NOMBRE	**OutSimulationPanel**	TIPO	Clase
PROPÓSITO	El propósito de este componente es agrupar todas las funcionalidades que permiten al usuario seguir el curso de las simulaciones dentro del sistema.		
FUNCIÓN	La función de este componente es monitorizar la simulación asociada a un determinado cloud de forma que el usuario pueda obtener información del transcurso de la misma.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	Este componente depende de la clase JPanel incluida en el paquete javax.swing de Java.		
INTERFACES	Ofrece dos interfaces: una es utilizada por el componente CV-001(GUI) para el envío de información del sistema, y la otra es utilizada por el componente CC-001(RequestManager) para el envío de información en tiempo real del transcurso de la simulación.		

Tabla 37: Componente Vista 006

IDENTIFICADOR	CV-007		
NOMBRE	**VisualPanel**	TIPO	Clase
PROPÓSITO	El propósito de este componente es agrupar todas las funcionalidades que están directamente relacionadas con la gestión de los resultados obtenidos en una determinada simulación.		
FUNCIÓN	La función de este componente consiste en mostrar al usuario información relacionada con los resultados obtenidos en una determinada simulación, así como permitir la opción de obtener un informe detallado de la misma en formato PDF.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	Este componente depende de la clase JPanel incluida en el paquete javax.swing de Java.		

INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CV-001(GUI), habilitada para el envío de información del sistema.

Tabla 38: Componente Vista 007

IDENTIFICADOR	**CV-008**		
NOMBRE	**PrincipalGraphPanel**	**TIPO**	Clase
PROPÓSITO	El propósito de este componente es recoger la funcionalidad asociada a la visualización de los resultados de una simulación de forma gráfica y detallada en el sistema.		
FUNCIÓN	La función de este componente consiste en representar gráficamente los resultados obtenidos directamente del simulador tras la realización de una ejecución del mismo.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	Este componente depende de la clase JPanel incluida en el paquete javax.swing de Java.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CV-006(VisualPanel) para el envío de información del sistema.		

Tabla 39: Componente Vista 008

IDENTIFICADOR	**CV-009**		
NOMBRE	**ThumbnailViewerPanel**	**TIPO**	Clase
PROPÓSITO	El propósito de este componente es recoger la funcionalidad asociada a la visualización de forma gráfica de todos los resultados obtenidos de una simulación en el sistema.		
FUNCIÓN	La función de este componente consiste en representar gráficamente todos los resultados obtenidos tras la realización de una simulación de un cloud y permitir la selección de uno u otro para su estudio.		

SUBORDINADOS	Este componente no tiene subordinados.
DEPENDENCIAS	Este componente depende de la clase JPanel incluida en el paquete javax.swing de Java.
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CV-006(VisualPanel) para el envío de información del sistema.

Tabla 40: Componente Vista 009

IDENTIFICADOR	**CV-010**		
NOMBRE	**ThumbnailPanel**	**TIPO**	Clase
PROPÓSITO	El propósito de este componente consiste en recoger la funcionalidad asociada a la representación de forma gráfica de los resultados de la simulación para su posterior visualización.		
FUNCIÓN	La función de este componente es recoger, tratar y transformar los resultados parciales de una simulación en gráficos de distinto tipo, que permitan al usuario su estudio y posterior almacenamiento.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	Este componente depende de la clase JPanel incluida en el paquete javax.swing de Java.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CV-008(ThumbnailViewerPanel) para el envío de información del sistema.		

Tabla 41: Componente Vista 010

IDENTIFICADOR	CV-011		
NOMBRE	**ButtonTabComponent**	TIPO	Clase
PROPÓSITO	El propósito de este componente es facilitar la gestión de los distintos paneles que se incluyen en la interfaz gráfica de usuario.		
FUNCIÓN	La función de este componente consiste en permitir al usuario una interacción más ágil con la aplicación gracias al empleo de pestañas independientes para la gestión de todos los componentes del sistema.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	Este componente depende de la clase JPanel incluida en el paquete javax.swing de Java.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente panel que se esté gestionando (desde CV-002 a CV-007) para el envío de información del sistema.		

Tabla 42: Componente Vista 011

Por otro lado, los componentes que forman la capa del controlador del sistema son los siguientes:

IDENTIFICADOR	CC-001		
NOMBRE	**RequestManager**	TIPO	Clase
PROPÓSITO	El propósito de este componente es tratar las solicitudes del usuario de la aplicación y trabajar con el modelo de datos del sistema para satisfacer dichas peticiones.		
FUNCIÓN	La función de este componente consiste en escuchar todas las peticiones procedentes de la interfaz de usuario y trabajar con los componentes del modelo de datos para satisfacer cada solicitud por separado.		
SUBORDINADOS	Este componente no tiene subordinados.		

DEPENDENCIAS	No tiene dependencias respecto a otros componentes.
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CV-001(GUI) para el envío de las peticiones de usuario.

Tabla 43: Componente Controlador 001

IDENTIFICADOR	**CC-002**		
NOMBRE	**ResultManager**	**TIPO**	Clase
PROPÓSITO	El propósito de este componente es agrupar todas las funcionalidades que tienen relación directa con los resultados obtenidos dada una determinada simulación en el sistema.		
FUNCIÓN	La función de este componente es agrupar todos los datos obtenidos como resultado de la simulación de un cloud en el sistema y tratarlos adecuadamente para la generación de informes gráficos.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CC-001(RequestManager) para el envío de los resultados obtenidos tras una simulación.		

Tabla 44: Componente Controlador 002

IDENTIFICADOR	**CC-003**		
NOMBRE	**FileSystem**	**TIPO**	Clase
PROPÓSITO	El propósito de este componente es agrupar todas las funcionalidades que tienen relación con la entrada y salida de información de archivos en el sistema.		
FUNCIÓN	La función de este componente consiste en comunicarse con el sistema de ficheros de la máquina para leer y/o escribir información en archivos que		

	permita abstraer funcionalidades como la creación de ficheros de configuración, el guardado de la configuración del usuario, la lectura de documentos gráficos para la generación del informe, etc.
SUBORDINADOS	Este componente no tiene subordinados.
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CC-001(RequestManager) para el envío de información de lectura/escritura en el sistema de ficheros de la máquina.

Tabla 45: Componente Controlador 003

IDENTIFICADOR	**CC-004**		
NOMBRE	**SimulationThread**	**TIPO**	Clase
PROPÓSITO	El propósito de este componente es agrupar todas las funcionalidades que tienen relación directa con la comunicación de la aplicación con el simulador, tanto para la introducción de parámetros de entrada como para la recogida de resultados de salida. Además, de este propósito principal, otro objetivo que se busca conseguir con el diseño de este componente es independizar el proceso de ejecución de la simulación del proceso principal de la aplicación.		
FUNCIÓN	La función de este componente es configurar los parámetros de entrada del simulador y lanzar la simulación deseada, permaneciendo al mismo tiempo a la escucha para la recogida de información en tiempo real. Este proceso será implementado como un *thread* o hilo independiente, de modo que el proceso principal de la aplicación no se verá afectado por la ejecución de las simulaciones.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.		

INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CC-003(FileSystem) para enviarle información del sistema.

Tabla 46: Componente Controlador 004

Para finalizar, la capa que contiene el modelo de datos del sistema está formada por los componentes descritos a continuación:

IDENTIFICADOR	**CM-001**		
NOMBRE	**VirtualMachine**	**TIPO**	Clase
PROPÓSITO	El propósito de este componente es agrupar todas las funcionalidades que están relacionadas con la gestión de cada máquina virtual dentro del sistema. Este componente busca abstraer cada máquina virtual en sí de los elementos que forman parte de ella.		
FUNCIÓN	La función de este componente es dar soporte a los componentes que forman parte de una determinada máquina virtual, estableciendo la visibilidad y el acceso a los mismos desde otros componentes.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CM-002 (VirtualMachineList) para el envío de información del sistema.		

Tabla 47: Componente Modelo 001

IDENTIFICADOR	CM-002		
NOMBRE	**VirtualMachineList**	TIPO	Clase
PROPÓSITO	El propósito de este componente es agrupar todas las funcionalidades que están relacionadas con la gestión de todo el conjunto de máquinas virtuales dentro del sistema. Este componente facilita las labores de los procesos que actúan sobre la totalidad de las máquinas que forman el sistema.		
FUNCIÓN	La función de este componente es permitir al usuario realizar acciones determinadas sobre el conjunto de máquinas virtuales del sistema, como pueden ser la búsqueda de máquinas dentro del sistema, la adición y el borrado de una máquina determinada, etc.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CC-001(RequestManager) para el envío de información del sistema.		

Tabla 48: Componente Modelo 002

IDENTIFICADOR	CM-003		
NOMBRE	**MachineType**	TIPO	Clase
PROPÓSITO	El propósito de este componente es representar los distintos tipos de máquinas virtuales que han sido añadidos en el sistema para su posterior vinculación con los clouds generados.		
FUNCIÓN	La función de este componente es dar soporte a los distintos tipos de máquinas virtuales que han sido incluidas previamente en la aplicación, estableciendo el acceso y la visibilidad a los mismos desde otros componentes del sistema.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.		

INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CC-001(RequestManager) para el envío de información del sistema.

Tabla 49: Componente Modelo 003

IDENTIFICADOR	**CM-004**		
NOMBRE	**Cloud**	**TIPO**	Clase
PROPÓSITO	El propósito de este componente es agrupar todas las funcionalidades que están relacionadas con la gestión de cada cloud dentro del sistema. Este componente busca abstraer cada cloud en sí mismo (como entidad) de los elementos que lo conforman.		
FUNCIÓN	La función de este componente es dar soporte a los componentes que forman parte de un determinado cloud, estableciendo cómo se gestionan los mismos y definiendo cómo es la visibilidad y el acceso estos desde otros componentes.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CC-001(RequestManager) para el envío de información del sistema.		

Tabla 50: Componente Modelo 004

IDENTIFICADOR	**CM-005**		
NOMBRE	**CloudList**	**TIPO**	Clase
PROPÓSITO	El propósito de este componente es agrupar todas las funcionalidades que están relacionadas con la gestión de todo el conjunto de clouds dentro del sistema. Este componente facilita las labores de los procesos que actúan sobre la totalidad de los clouds que forman el sistema.		
FUNCIÓN	La función de este componente es permitir al usuario realizar acciones determinadas sobre el conjunto de clouds del sistema, como pueden ser la		

	búsqueda de clouds dentro del sistema, la adición y el borrado de un determinado cloud, etc.
SUBORDINADOS	Este componente no tiene subordinados.
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CC-001(RequestManager) para el envío de información del sistema.

Tabla 51: Componente Modelo 005

IDENTIFICADOR	**CM-006**		
NOMBRE	**Combina**	**TIPO**	Clase
PROPÓSITO	El propósito de este componente es representar cada uno de los elementos (nodos) que forman un determinado cloud dentro del sistema y abstraer los elementos internos del resto de la aplicación.		
FUNCIÓN	La función de este componente es dar soporte a cada elemento que forma parte de un determinado cloud dentro del sistema, estableciendo cómo se gestionan sus componentes y definiendo la visibilidad y el acceso a los mismos desde otros componentes de la aplicación.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CM-004 (Cloud) para el envío de información del sistema.		

Tabla 52: Componente Modelo 006

IDENTIFICADOR	CM-007		
NOMBRE	**Application**	TIPO	Clase
PROPÓSITO	El propósito de este componente es agrupar todas las funcionalidades que están relacionadas con la gestión de cada aplicación dentro del sistema. Este componente busca abstraer cada aplicación en sí misma (como entidad) de los elementos que la conforman.		
FUNCIÓN	La función de este componente es dar soporte a los parámetros que forman parte de la configuración de una determinada aplicación, estableciendo cómo se gestionan las mismas y definiendo cómo es la visibilidad y el acceso estos desde otros componentes.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CM-012(AppList) para el envío de información del sistema.		

Tabla 53: Componente Modelo 007

IDENTIFICADOR	CM-008		
NOMBRE	**ApplicationType**	TIPO	Clase
PROPÓSITO	El propósito de este componente consiste en dar soporte a cada tipo de aplicación dentro del sistema y gestionar sus valores propios. Esta clase es la que permite que la carga de aplicaciones sea dinámica.		
FUNCIÓN	La función de este componente es dar soporte a la definición de atributos o características de cada aplicación, estableciendo cuales son los parámetros propios de ese tipo de aplicación y definiendo cómo es el acceso y la visibilidad a los mismos desde otros componentes del sistema.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	Este componente depende del componente CM-007(Application), del cual hereda su comportamiento.		

INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CM-007 (Application) para el envío de información del sistema.

Tabla 54: Componente Modelo 008

IDENTIFICADOR	**CM-009**		
NOMBRE	**UserJob**	**TIPO**	Clase
PROPÓSITO	El propósito de este componente consiste en representar una determinada entrada dentro de una tarea o trabajo definido por un usuario. Una tarea de usuario puede incluir un conjunto de aplicaciones, donde cada una se ejecute sobre un número determinado de maquinas virtuales. La relación aplicación – maquinas virtuales se representa mediante un objeto de esta misma clase.		
FUNCIÓN	La función de este componente es dar soporte a cada una de las entradas de una determinada tarea de usuario, estableciendo cuales son los parámetros propios de cada una de ellas y definiendo cómo es el acceso y la visibilidad a las mismas desde otros componentes del sistema.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CM-010 (User) para el envío de información del sistema.		

Tabla 55: Componente Modelo 009

IDENTIFICADOR	**CM-010**		
NOMBRE	**User**	**TIPO**	Clase
PROPÓSITO	El propósito de este componente consiste en representar una determinada tarea o trabajo de usuario dentro del sistema y gestionar sus valores propios, todo ello con el fin de su posterior utilización en la configuración de los futuros clouds a generar.		

FUNCIÓN	La función de este componente es dar soporte a las tareas de usuario que son insertadas en la aplicación, estableciendo cuales son las aplicaciones que se quieren ejecutar y el conjunto de maquinas sobre el que se desean ejecutar, y definiendo cómo es el acceso y la visibilidad a los mismos desde otros componentes del sistema.
SUBORDINADOS	Este componente no tiene subordinados.
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CM-011 (UserList) para el envío de información del sistema.

Tabla 56: Componente Modelo 010

IDENTIFICADOR	**CM-011**		
NOMBRE	**UserList**	**TIPO**	Clase
PROPÓSITO	El propósito de este componente es agrupar todas las funcionalidades que están relacionadas con la gestión de todo el conjunto de tareas dentro del sistema. Este componente facilita las labores de los procesos que actúan sobre la totalidad de las tareas de usuario incluidas en el sistema.		
FUNCIÓN	La función de este componente es permitir al usuario realizar acciones determinadas sobre el conjunto de tareas del sistema, como pueden ser la búsqueda de tareas dentro del sistema, la adición y el borrado de un trabajo determinado, etc.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CC-001(RequestManager) para el envío de información del sistema.		

Tabla 57: Componente Modelo 011

IDENTIFICADOR	CM-012		
NOMBRE	**AppList**	TIPO	Clase
PROPÓSITO	El propósito de este componente es agrupar todas las funcionalidades que están relacionadas con la gestión de todo el conjunto de aplicaciones dentro del sistema. Este componente facilita las labores de los procesos que actúan sobre la totalidad de las aplicaciones incluidas en el sistema.		
FUNCIÓN	La función de este componente es permitir al usuario realizar acciones determinadas sobre el conjunto de aplicaciones del sistema, como pueden ser la búsqueda de aplicaciones dentro del sistema, la adición y el borrado de una aplicación determinada, etc.		
SUBORDINADOS	Este componente no tiene subordinados.		
DEPENDENCIAS	No tiene dependencias respecto a otros componentes.		
INTERFACES	Ofrece una interfaz, la cual es requerida por el componente CC-001(RequestManager) para el envío de información del sistema.		

Tabla 58: Componente Modelo 012

Una vez que se han identificado todos y cada uno de los componentes que forman parte del sistema a desarrollar, a continuación se describirán detalladamente algunas de las principales funcionalidades que se implementarán en el sistema y que formarán la base de algunos de los casos de uso expuestos en capítulos anteriores.

Estas descripciones de funcionalidades incluyen datos muy interesantes como los parámetros de entrada y el valor de retorno de dichas funciones pero sobre todo resulta interesante comprobar el flujo de ejecución que sigue la información hasta que se obtiene el resultado deseado (el pseudocódigo).

La mayoría de las funcionalidades aquí descritas pertenecen a la clase *RequestManager*, ya que es la encargada de gestionar las peticiones del cliente y trabajar con el modelo de datos para satisfacer dichas necesidades. Así pues, las funcionalidades a detallar elegidas son las siguientes:

NOMBRE	**DeleteMachine**	TIPO	Método
COMPONENTE	RequestManager.java		
DESCRIPCIÓN	Este método es invocado desde la vista del sistema (componente GUI) en el momento en el que el usuario elige la opción de eliminar una determinada máquina virtual del sistema. Su objetivo principal consiste en eliminar una determinada máquina virtual del sistema y borrar a su vez todas las referencias a dicha máquina en los distintos clouds, es decir, borrar el tipo de máquina asociado (componente *MachineType*).		
VALOR DE RETORNO	*True* si el proceso ha finalizado con éxito y *false* en caso contrario.		
PARÁMETROS	• Nombre de la máquina virtual a borrar.		

PSEUDOCÓDIGO	• Primero se borra la máquina virtual en sí. Así, se recorre la lista de máquinas virtuales de la aplicación (componente VirtualMachineList) y para cada máquina:
	  ○ Se comprueba si el nombre es igual al parámetro y, en caso afirmativo, se borra de la lista.
	• Después se elimina la referencia a dicha máquina en cada uno de los clouds. Para ello, se recorre la lista de clouds (componente CloudList) y para cada cloud:
	  ○ Se recorren todas sus entradas (componente Combina) y para cada uno de ellos:
	  ▪ Si el tipo de máquina definida es la misma que la que acabamos de borrar, se elimina dicha entrada completamente del cloud.
	• Por último, se elimina el tipo de máquina asociado a la máquina virtual eliminada. Para ello se recorre la lista de instancias MachineType y para cada una de ellas:
	  ○ Si su nombre es igual al de la máquina eliminada, se borra completamente de la lista.		

Tabla 59: Método DeleteMachine

NOMBRE	**OpenAppConfiguration**	TIPO	Método
COMPONENTE	RequestManager.java		
DESCRIPCIÓN	Este método es invocado desde la vista del sistema (componente GUI) en el momento en el que el usuario inicia una nueva sesión en la aplicación. Tiene dos objetivos principales: por un lado, lee de disco todas las aplicaciones almacenadas y las instancia en el sistema, y por otro devuelve el nombre de todas estas aplicaciones almacenadas en la BBDD para que se muestren en la interfaz.		
VALOR DE RETORNO	Array de nombres de las aplicaciones leídas.		
PARÁMETROS	No tiene parámetros de entrada.		

PSEUDOCÓDIGO	• Se inicializa un array de nombres.
	• Se lee el directorio por defecto en el que se almacenan todas las aplicaciones del sistema y se devuelve un array con las rutas absolutas a cada aplicación.
	• Para cada ruta encontrada:
	  ○ Se llama a un método “deserializar” y se le pasa la ruta, obtiendo una nueva instancia del tipo de aplicación que hubiera almacenada.
	  ○ Se añade dicha tarea a la lista de aplicaciones del sistema (componente *AppList*).
	  ○ Se añade el nombre de la aplicación al array de nombres principal.
	• Se devuelve el array con los nombres de todas las rutas a la vista de la aplicación para que ésta se la muestre al usuario.		

Tabla 60: Método OpenAppConfiguration

NOMBRE	**SaveConfiguration**	TIPO	Método
COMPONENTE	RequestManager.java		
DESCRIPCIÓN	Este método es invocado desde la vista del sistema (componente GUI) en el momento en el que el usuario elige la opción de guardar la configuración personal. El objetivo del método es guardar toda la información relativa a los elementos principales del sistema manteniendo la consistencia del mismo. Por ello, antes de realizar el almacenamiento se limpia la información anterior y se guarda la información a distintos ficheros de disco.		
VALOR DE RETORNO	*True* si el proceso ha finalizado con éxito y *false* en caso contrario.		
PARÁMETROS	No tiene parámetros de entrada.		

PSEUDOCÓDIGO	• Se llama al componente que gestiona el sistema de ficheros (componente *FileSystem*) y se elimina toda la información del repositorio.
	• Para cada máquina virtual:
	  ○ Se obtiene la máquina virtual de la posición X de la lista de máquinas virtuales (componente *VirtualMachineList*).
	  ○ Se crea un fichero en la ruta adecuada, identificado con el nombre de la propia máquina virtual.
	  ○ Se serializa la máquina virtual y se escribe el resultado en dicho fichero.
	• Para cada cloud se actúa de forma análoga a la expuesta en el punto anterior.
	• Para cada tarea se actúa de forma análoga a la expuesta en el punto anterior.
	• Para cada aplicación se actúa de forma análoga a la expuesta en el punto anterior.
	• Se devuelve el resultado de la operación a la vista para que se le notifique al usuario el éxito de la operación.		

Tabla 61: Método SaveConfiguration

NOMBRE	**ExecuteSimulation**	TIPO	Método
COMPONENTE	FileSystem.java		
DESCRIPCIÓN	Este método es invocado por el gestor de peticiones de la aplicación (componente *RequestManager*) en el momento en el que el usuario elige la opción de simular un determinado cloud del sistema. El objetivo del método consiste en comunicarse con el simulador para lanzar una determinada ejecución en el mismo. Este método es llamado después de que se hayan creado todos los ficheros de configuración que necesita el simulador.		
VALOR DE RETORNO	No tiene un valor de retorno definido ya que el estado de la acción se monitoriza en todo momento.		
PARÁMETROS	• Cloud que se desea simular • Área de texto en la que se desea monitorizar la salida del simulador (perteneciente a una instancia del componente *OutSimulationPanel*).		
PSEUDOCÓDIGO	• Se crea una nueva instancia de la clase *SimulationThread*, pasándole como parámetros la instancia del cloud a simular y el componente área de texto. • Esta clase *SimulationThread* implementa la interfaz *Runnable*, por lo que se puede ejecutar su contenido como un hilo independiente del proceso principal. Por ello, creamos un nuevo *thread*, le pasamos la instancia antes creada y comenzamos su ejecución llamando al método *start*(). • Una vez toma el control el hilo, se crea una instancia de la clase ProcessBuilder (sirve para lanzar procesos al sistema operativo) y se le dice que ejecute el contenido de un fichero donde están los comandos apropiados. • Para definir las librerías que necesita el simulador se utiliza un objeto de la clase Map, y se pasan las rutas donde se encuentran cada una de ellas. • Se lanza el proceso y se redirige su salida a un buffer. • Durante el resto del proceso, se lee una línea del buffer y mientras		

	contenga datos: ○ Se almacena esta línea en el área de texto que se pasó por parámetro, de forma que el usuario lea en su interfaz el resultado. ○ Se lee la siguiente línea. • Finalmente, se cierran todos los flujos de datos.

Tabla 62: Método CreateSimulation

NOMBRE	**GenerateGraph**	TIPO	Método
COMPONENTE	ResultManager.java		
DESCRIPCIÓN	Este método es invocado desde la vista del sistema (componente GUI) en el momento en el que el usuario elige la opción de visualizar los resultados obtenidos después de que haya finalizado correctamente la simulación de un cloud en el sistema. Cuando ha finalizado un proceso de simulación dentro del sistema, aparecerá un indicador en la interfaz de usuario notificando dicho evento. Si el usuario decide ver los resultados de la simulación, interactuará con dicho indicador y se llevarán a cabo dos acciones consecutivas. En primer lugar, se leerá el fichero de salida generado por el simulador y se almacenarán los datos en las estructuras del propio componente ResultManager. Después de esto, se realizará una llamada al método GenerateGraph(), que interpretará esta información almacenada y generará los gráficos oportunos de acuerdo a la información leída.		
VALOR DE RETORNO	*True* si el proceso ha finalizado con éxito y *false* en caso contrario.		
PARÁMETROS	• Nombre del cloud que ha sido simulado. • Lista de máquinas virtuales del sistema. • Lista de aplicaciones del sistema.		
PSEUDOCÓDIGO	• Se comprueba si el directorio en el que se deben guardar las gráficas existe, y en caso de que no exista se crea. Si existe dicho directorio		

	se elimina toda la información que hubiera previamente almacenada en él. • Se recorre la estructura de resultados de esta clase. Según el formato del fichero de salida del simulador, existe una línea por cada trabajo de usuario que ha sido ejecutado en cada máquina virtual (se recoge la tarea ejecutada, el tiempo empleado,…) • Se trata la información que se almacena en los parámetros de la clase ResultManager (un array por cada resultado de ejecución de un trabajo en una máquina virtual determinada) y se generan las gráficas oportunas, utilizando la biblioteca de ayuda *jfreechart.* • Finalmente, se guarda en ficheros JPEG todas las gráficas generadas, para su posterior utilización (o bien mostrándolas al usuario en la GUI o bien incluyéndolas en el informe de la ejecución).

Tabla 63: Método GenerateGraph

NOMBRE	**AddCloudToList**	TIPO	Método
COMPONENTE	ResultManager.java		
DESCRIPCIÓN	Este método es invocado desde la vista del sistema (componente GUI) en el momento en el que el usuario elige la opción de aplicar los cambios realizados cuando está dando de alta o bien editando un cloud anteriormente creado en el sistema. Su objetivo, dependiendo de la acción a realizar, será crear una nueva instancia de la clase cloud con los atributos que el usuario haya definido en la interfaz del sistema, o bien modificar los atributos del propio cloud en el caso de que estemos editando un cloud anteriormente creado en la aplicación.		
VALOR DE RETORNO	*True* si el proceso ha finalizado con éxito y *false* en caso contrario.		
PARÁMETROS	• Nombre del cloud. • Estructura compleja de arrays con los elementos que deben formar parte del cloud.		

	• Opción a ejecutar: *add* para crear un nuevo cloud y *edit* para modificar un cloud existente.

PSEUDOCÓDIGO	• Se comprueba la opción a ejecutar: si se ha elegido crear un cloud nuevo:
	  ○ Se llama al método *GetCloudByName*() de la lista de clouds para comprobar que no haya ningún cloud ya creado con el mismo nombre.
	  ○ Si hay alguno, se devuelve un mensaje de error al usuario.
	  ○ Si no existe ninguno nos disponemos a crear el nuevo cloud. Por cada subestructura del array pasado por parámetro se crea una instancia del componente *Combina*, se genera el cloud con toda esta información como atributos y se añade finalmente a la lista de clouds del sistema (componente *CloudList*).
	• Si la opción elegida ha sido la de editar un cloud existente:
	  ○ Se modifica el cloud seleccionado, para lo que se llama al método *editCloud*() del componente *CloudList*. Este método se encarga de borrar los elementos que contiene el cloud (lista de instancias del componente *Combina*) y añade los nuevos en base a la información pasada por parámetro.
	• Finalmente, se devuelve un valor u otro dependiendo del resultado de las operaciones realizadas.

Tabla 64: Método AddCloudToList

4.5 Sintaxis del fichero de resultados

En este apartado se muestra la sintaxis del fichero que es generado por el propio simulador iCanCloud, después de una ejecución. Como se puede apreciar en la siguiente figura, el contenido de este fichero está formado por una serie de líneas que contienen información relevante de la simulación realizada.

A su vez, cada una de estas líneas está formada por una serie de parámetros separados una serie de delimitadores. Estos parámetros son los siguientes:

- **Token**: identificador del trabajo o tarea de usuario que ha sido ejecutado **en una máquina virtual**. Se trata de una cadena de caracteres formada por la unión de varios elementos separados por el delimitador '$'. Estos subelementos identifican a:
    - Tarea de usuario: corresponde con el número de secuencia de la tarea de trabajo a la que pertenece la aplicación que se ha ejecutado.
    - Aplicación: corresponde con el número de secuencia de la aplicación que se ha ejecutado dentro de la tarea definida anteriormente.
    - Iteración: corresponde a la iteración que se ha ejecutado de la aplicación en la máquina virtual asignada.

  Así, por ejemplo, el token "1$3$1" haría referencia a los resultados obtenidos (de atrás hacia adelante) de la ejecución de la primera iteración, de la tercera aplicación definida en el primer trabajo que debía ejecutar el cloud en cuestión.

  Este elemento se distingue de los demás (dentro de cada línea del fichero) mediante la separación con el delimitador "**:**".
- **Tiempo de E/S**: periodo de tiempo que ha estado la aplicación realizando operaciones de entrada y salida (lectura y escritura en fichero).
- **Tiempo de CPU**: periodo de tiempo que se tarda en ejecutar todas las instrucciones de la aplicación en el procesador de una máquina virtual.
- **Tiempo real de simulación**: tiempo que ha tardado una aplicación en ejecutarse en el simulador.
- **Tiempo simulado**: periodo de tiempo que se emplearía en ejecutar la aplicación en un entorno no simulado, de acuerdo a los datos obtenidos en la simulación.

Así pues, un ejemplo de fichero de salida del simulador es el que aparece reflejado en la Figura 47 que aparece a continuación:

Results.txt

```
1$1$1:0.00797013#2#0#2.00797
1$1$1:0.00797013#2#0#2.00797
1$2$1:0.00797013#2#0#2.00797
1$2$1:0.00797013#2#0#2.00797
1$2$1:0.00797013#2#0#2.00797
1$2$1:0.00797013#2#0#2.00797
1$2$1:0.00797013#2#0#2.00797
1$2$1:0.00797013#2#0#2.00797
1$3$1:0.00797013#2#0#2.00797
1$3$1:0.00799929#2#0#2.008
1$3$1:0.00799929#2#0#2.008
1$3$2:0.00799929#2#0#2.008
1$3$2:0.00799929#2#0#2.008
1$3$2:0.00799929#2#0#2.008
1$4$1:0.00799929#2#0#2.008
1$4$2:0.00799929#2#0#2.008
```

Figura 47: Fichero de salida del simulador iCanCloud

Observando la salida se pueden llegar a una serie de conclusiones sobre la arquitectura empleada en el cloud simulado. En la Figura 47 se puede observar cómo el cloud simulado está formado por un solo trabajo de usuario, que a su vez está compuesto por 4 entradas distintas (debe ejecutar 4 aplicaciones en distintas máquinas virtuales), donde:

- La primera aplicación (1$1), tiene una sola iteración (1$1$1) y se ejecuta en dos máquinas virtuales (por ello tenemos dos valores), lo cual explica que tengamos dos líneas con el mismo token.
- La segunda aplicación (1$2), tiene una sola iteración (1$2$1), pero esta vez se ejecuta en 6 máquinas distintas, por ello se pueden contar 6 entradas del fichero con el mismo token (cada una asociada al resultado obtenido en cada máquina).
- La tercera aplicación (1$3), tiene dos iteraciones, es decir se ejecuta dos veces (1$3$1 y 1$3$2) sobre 3 máquinas virtuales distintas.
- Por último, la cuarta iteración (1$4) tiene también dos iteraciones (1$4$1 y 1$4$2) y se ejecuta sobre una sola máquina virtual.

El sistema deberá leer esta información e interpretarla de manera adecuada para la creación de informes y de gráficas que aporten datos concluyentes al usuario de la herramienta.

4.6 Sintaxis de los ficheros de almacenamiento

A continuación se procede a describir el formato de los ficheros de almacenamiento de información que generará la herramienta a desarrollar. Estos ficheros de almacenamiento no son sino aquellos en los que se van a recoger los datos asociados a los componentes creados en el sistema, es decir, las máquinas virtuales, las tareas, las aplicaciones y los clouds deseados.

Aunque existen diversas sintaxis posibles que nos permitirían almacenar la información en disco, muchas de ellas dependen directamente del lenguaje de programación utilizado en la implementación de la propia herramienta. Así pues, se ha decidido utilizar un estándar para el intercambio de información estructurada como es el lenguaje **XML** (eXtensible Markup Language), uno de los más utilizados en la actualidad.

Como se comentó en apartados anteriores, la herramienta va a trabajar con varios elementos fundamentales, cuya información deberá almacenar y recuperar fácilmente: las máquinas virtuales, las tareas, las aplicaciones y los clouds. Cada uno de estos elementos se gestionará por separado en la herramienta por lo que su almacenamiento también será independiente. Así, nos encontramos con que existe un conjunto de cuatro tipos distintos de archivos de almacenamiento.

Además, para facilitar el acceso y recuperación de esta información, se ha decidido gestionar estos archivos en distintos directorios, de acuerdo al tipo de archivo que contenga cada uno de ellos.

La Figura 48 que aparece a continuación muestra un ejemplo de un archivo XML que contiene la información de una máquina virtual del sistema.

Como se puede apreciar, los atributos simples de una determinada entidad aparecen representados por la etiqueta *void property* seguido del nombre del atributo, del tipo del atributo y del propio valor.

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0_13" class="java.beans.XMLDecoder">
 <object class="pfcpani.model.VirtualMachine">
  <void property="coreNumber">
   <int>4</int>
  </void>
  <void property="coreSpeed">
   <int>5</int>
  </void>
  <void property="diskCapacity">
   <int>4</int>
  </void>
  <void property="diskNumber">
   <int>4</int>
  </void>
  <void property="diskSpeed">
   <int>1</int>
  </void>
  <void property="diskWrite">
   <int>1</int>
  </void>
  <void property="machineName">
   <string>high</string>
  </void>
  <void property="memorySize">
   <int>4000000</int>
  </void>
 </object>
</java>
```

Figura 48: Fichero XML de almacenamiento de una máquina virtual

A continuación se muestra un ejemplo de un fichero típico de una aplicación almacenada en el sistema. La Figura 49 muestra un archivo en el que se serializa la información asociada a una determinada aplicación. Como se puede apreciar, se trata de una aplicación de tipo LocalApplication (ya se comentó en apartados anteriores en qué consistía este tipo de tarea) como indica la línea:

<object class="pfcpani.model.ApplicationType">

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0_13" class="java.beans.XMLDecoder">
 <object class="pfcpani.model.Application">
  <void property="appName">
   <string>FirstApp</string>
  </void>
 <object class="pfcpani.model.ApplicationType">
  <void property="name">
   <string>LocalApplication</string>
  </void>
  <vars>
  <var>
  <varName>application_netType</varName>
  <varType>string</varType>
  </var>
  <var>
  <varName>startDelay</varName>
  <varType>double</varType>
  </var>
  </vars>
 </object>
  <values>
   <value>BASIC</value>
   <value>50</value>
  </values>
 </object>
</java>
```

Figura 49: Fichero XML de almacenamiento de una aplicación del sistema

Esta aplicación se ha denominado en el sistema como *fistApp*, y entre sus parámetros configurables se encuentran, además del tipo (clase *ApplicationType*) que indica que es *localApplication*, otras como el tipo de red (application_netType).

Las aplicaciones, como se comento antes, se leen dinámicamente en el sistema. Así, pueden surgir nuevos tipos de aplicaciones con distinto número y tipo de parámetros que otras. Por ello, no hay una estructura fija de serialización en fichero XML.

Para almacenar los parámetros se utilizan principalmente dos etiquetas: la etiqueta <vars> contiene tantas subetiquetas como parámetros tiene el tipo de aplicación leída. Cada una de estas subetiquetas contiene a su vez el nombre del atributo y el tipo de atributo (en las

etiquetas <varName> y <varType> respectivamente). Por otro lado, la etiqueta <values> recoge los valores ordenados de cada uno de los atributos antes descritos.

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0_13" class="java.beans.XMLDecoder">
 <object class="pfcpani.model.User">
  <void property="userName">
   <string>Gabriel</string>
  </void>
  <object class="pfcpani.model.UserJob">
   <void property="numVms">
    <int>3</int>
   </void>
   <void property="typeVm">
    <String>medium</String>
   </void>
   <void property="typeJob">
    <String>SecondApp</String>
   </void>
   <void property="numJobs">
    <int>7</int>
   </void>
  </object>
  <object class="pfcpani.model.UserJob">
   <void property="numVms">
    <int>6</int>
   </void>
   <void property="typeVm">
    <String>high</String>
   </void>
   <void property="typeJob">
    <String>FirstApp</String>
   </void>
   <void property="numJobs">
    <int>4</int>
   </void>
  </object>
 </object>
</java>
```

Figura 50: Fichero XML de almacenamiento de una tarea del sistema

La Figura 50 muestra un ejemplo de fichero XML en el que se ha guardado la configuración de una determinada tarea de usuario. Como se puede apreciar, la propiedad *userName* determinada el nombre de la tarea, en este caso se denomina “Gabriel” para identificar los trabajos que quiere lanzar dicho usuario.

Esta tarea va a ejecutar dos aplicaciones distintas, por lo que tiene dos etiquetas

<object class="pfcpani.model.UserJob">

distintas. La primera aplicación a ejecutar será una de tipo *SecondApp*, que se ejecutara sobre 3 maquinas virtuales de tipo *medium*. Este proceso se repetirá 7 veces. Por otro lado, la otra aplicación a ejecutar es una de tipo *FistApp*, la cual se ejecutara sobre 6 maquinas virtuales de tipo *high*, proceso que se repetirá un total de 4 veces.

De esta forma se puede apreciar que con la estructura del fichero XML anterior tenemos todo lo necesario para almacenar en el sistema la información asociada a las tareas que un usuario desea ejecutar en el sistema.

La Figura 51 que aparece a continuación muestra el contenido de un fichero de almacenamiento en el que se ha serializado la información asociada a un cloud. En la ilustración puede observarse cómo, además de los atributos simples, aparecen las subentidades que forman parte de otras entidades principales mediante la utilización de la etiqueta *object class* seguida del nombre de la entidad (en este ejemplo, el cloud almacenado está formado por dos objetos *Combina*, donde cada uno de ellos tiene un objeto *MachineType*).

Cada objeto Combina representa así una entrada determinada de la infraestructura ofrecida por el proveedor del cloud. De esta manera, determina el numero de maquinas de un determinado tipo que se contratan para la ejecución de las tareas del usuario.

Cada entrada tiene una serie de atributos simples: el número de máquinas virtuales (propiedad elements), el coste de cada máquina (propiedad cost), y el tipo de maquina contratada (objeto MachineType).

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0_13" class="java.beans.XMLDecoder">
 <object class="pfcpani.model.Cloud">
  <void property="cloudName">
   <string>TestCloud</string>
  </void>
  <object class="pfcpani.model.Combina">
   <void property="elements">
    <int>300</int>
   </void>
   <void property="cost">
    <double>7.0</double>
   </void>
   <object class="pfcpani.model.MachineType">
    <void property="name">
     <string>medium</string>
    </void>
   </object>
  </object>
  <object class="pfcpani.model.Combina">
   <void property="elements">
    <int>2</int>
   </void>
   <void property="cost">
    <double>24.0</double>
   </void>
   <object class="pfcpani.model.MachineType">
    <void property="name">
     <string>high</string>
    </void>
   </object>
  </object>
 <object class="pfcpani.model.User">
  <void property="userName">
   <string>Cana</string>
  </void>
  <object class="pfcpani.model.UserJob">
   <void property="numVms">
    <int>3</int>
   </void>
   <void property="typeVm">
```

```
   <String>high</String>
  </void>
  <void property="typeJob">
   <String>FirstApp</String>
  </void>
  <void property="numJobs">
   <int>2</int>
  </void>
 </object>
 <object class="pfcpani.model.UserJob">
  <void property="numVms">
   <int>1</int>
  </void>
  <void property="typeVm">
   <String>high</String>
  </void>
  <void property="typeJob">
   <String>FirstApp</String>
  </void>
  <void property="numJobs">
   <int>1</int>
  </void>
 </object>
</object>
<object class="pfcpani.model.User">
 <void property="userName">
  <string>Gabriel</string>
 </void>
 <object class="pfcpani.model.UserJob">
  <void property="numVms">
   <int>3</int>
  </void>
  <void property="typeVm">
   <String>medium</String>
  </void>
  <void property="typeJob">
   <String>SecondApp</String>
  </void>
  <void property="numJobs">
   <int>7</int>
  </void>
```

```
 </object>
 <object class="pfcpani.model.UserJob">
  <void property="numVms">
   <int>6</int>
  </void>
  <void property="typeVm">
   <String>high</String>
  </void>
  <void property="typeJob">
   <String>FirstApp</String>
  </void>
  <void property="numJobs">
   <int>4</int>
  </void>
 </object>
</object>
</object>
</java>
```

Figura 51: Fichero XML de almacenamiento de un cloud

4.7 Sintaxis de los ficheros de configuración

En este apartado, se van a describir brevemente cuales son los ficheros de configuración que debe generar la herramienta de forma automática para permitir la simulación de un cloud. El simulador con el que se comunicará este sistema recibe una serie de ficheros de configuración de entrada que son los siguientes:

- Fichero **(Nombre del cloud).cfg** (por ejemplo “TestCloud.cfg”).
- **Run**
- **Ior.txt**
- **PreLoadFiles.txt**
- **NetFeatures.ini**
- **Omnetpp.ini**
- **Scenario.ned**
- **Ips.txt**

Estos ficheros contienen información de configuración del cloud además de otros datos que necesita el simulador para ejecutar. Veamos el contenido de estos ficheros.

TestCloud.cfg

```
P#medium:300
P#high:20
/
U#1
V#high:3
J#1$1$1%LocalApplication:string application_netType BASIC ;double startDelay 50 ;
V#high:3
J#1$1$2%LocalApplication:string application_netType BASIC ;double startDelay 50 ;
V#high:1
J#1$2$1%LocalApplication:string application_netType BASIC ;double startDelay 50 ;
@
U#2
V#medium:3
J#2$1$1%LocalApplication:string application_netType BASIC ;double startDelay 100 ;
V#medium:3
J#2$1$2%LocalApplication:string application_netType BASIC ;double startDelay 100 ;
V#medium:3
J#2$1$3%LocalApplication:string application_netType BASIC ;double startDelay 100 ;
V#medium:3
J#2$1$4%LocalApplication:string application_netType BASIC ;double startDelay 100 ;
V#medium:3
J#2$1$5%LocalApplication:string application_netType BASIC ;double startDelay 100 ;
V#medium:3
J#2$1$6%LocalApplication:string application_netType BASIC ;double startDelay 100 ;
V#medium:3
J#2$1$7%LocalApplication:string application_netType BASIC ;double startDelay 100 ;
V#high:6
J#2$2$1%LocalApplication:string application_netType BASIC ;double startDelay 50 ;
V#high:6
J#2$2$2%LocalApplication:string application_netType BASIC ;double startDelay 50 ;
V#high:6
J#2$2$3%LocalApplication:string application_netType BASIC ;double startDelay 50 ;
V#high:6
J#2$2$4%LocalApplication:string application_netType BASIC ;double startDelay 50 ;
@
```

Figura 52: Fichero de configuración TestCloud.cfg

En la Figura 52 aparece un ejemplo de fichero *.cfg*, en el que se definen en la parte superior el total de máquinas virtuales que hay disponibles en el cloud, y a continuación se definen cada uno de los trabajos o tareas de usuario, donde para cada aplicación incluida se determinan el número y tipo de máquinas en las que se va a ejecutar, además de los parámetros de la propia aplicación.

run

```
#!/bin/sh
../../src/run_iCanCloud $*
```

Figura 53: Fichero de ejecución run

ior.txt

```
1
FS:/:0
```

Figura 54: Fichero de configuración ior.txt

preLoadFiles.txt

```
2
/input.dat:1000000
/output.dat:1000000
```

Figura 55: Fichero de configuración preLoadFiles.txt

En la Figura 53 anterior se muestra el fichero que supone el inicio de una simulación cualquiera. Por su parte, en la Figura 54 y en la Figura 55 se muestran dos archivos de configuración del simulador, que si bien no contienen información exclusiva del cloud, son necesarios para la correcta simulación del mismo.

Por su parte, en la Figura 56 que aparece a continuación se muestra un ejemplo de fichero netFeatures.ini, el cual recoge información de red como los parámetros de la comunicación TCP, IP o ARP entre las distintas máquinas (véase Anexo III – Glosario de acrónimos y abreviaturas para más información).

netFeatures.ini

```
### MAC settings
**.mac.address = "auto"
**.mac[*].address = "auto"
##########################################################################################
### Switch
**.relayUnitType = "MACRelayUnitPP"
##########################################################################################
###_TCP_settings
**.tcp.mss = 1024                                    # Maximum Segment Size
**.tcp.advertisedWindow = 14336                      # 14*mss
**.tcp.tcpAlgorithmClass = "TCPReno"                 # TCP Algorithm
**.tcp.sendQueueClass = "TCPMsgBasedSendQueue"       # TCP sendQueue
**.tcp.receiveQueueClass = "TCPMsgBasedRcvQueue"     # TCP Receive queue
**.tcp.recordStats = false                           # No record scalars
##########################################################################################
###_IP_module
**.ip.procDelay = 1us
##########################################################################################
###_ARP_configuration
**.arp.retryTimeout = 1s
**.arp.retryCount = 3
**.arp.cacheTimeout = 100s
##########################################################################################
###_NIC_configuration
**.ppp[*].queueType = "DropTailQueue" # in routers
**.ppp[*].queue.frameCapacity = 10000
##########################################################################################
###_Network_interface_configuration
**.eth[*].queueType = "DropTailQueue"
**.eth[*].queue.frameCapacity = 10000
**.eth[*].mac.promiscuous = false
**.eth[*].mac.writeScalars = false
**.eth[*].encap.writeScalars = false
##########################################################################################
###_nam_trace
**.nam.logfile = ""
**.nam.prolog = ""
##########################################################################################
```

Figura 56: Fichero de configuración netFeatures.ini

omnetpp.ini

```
[General]
network = TestCloud
tkenv-plugin-path = ../../../etc/plugins
ned-path = ../../../inet/src
### Number of VM of the provider
TestCloud.numVM_medium = 300
TestCloud.numVM_high = 20

##########################################################################
###     Definition of node vm_medium[*]
##########################################################################
### CPU system
TestCloud.vm_medium[*].cpuModule.cpuCoreType = "CPUcore"
TestCloud.vm_medium[*].cpuModule.CPUcore[*].speed = 4
TestCloud.vm_medium[*].cpuModule.CPUcore[*].tick_s = 0.001
### Storage system
TestCloud.vm_medium[*].bsModule[*].diskType = "Disk_400GB_LI"
TestCloud.vm_medium[*].bsModule[*].cacheType = "NullCache"
### Operating System
TestCloud.vm_medium[*].osModule.networkServiceType = "NetworkService"
TestCloud.vm_medium[*].osModule.cpuSchedulerType = "CPU_Scheduler_FIFO"
TestCloud.vm_medium[*].osModule.memoryType = "BasicMainMemory"
### CPU Scheduler
TestCloud.vm_medium[*].osModule.cpuScheduler.numCPUs = 2
### Memory
TestCloud.vm_medium[*].osModule.memory.readLatencyTime_s = 1
TestCloud.vm_medium[*].osModule.memory.writeLatencyTime_s = 1
TestCloud.vm_medium[*].osModule.memory.size_KB = 4000000
TestCloud.vm_medium[*].osModule.memory.blockSize_KB = 128
### Volume Manager
TestCloud.vm_medium[*].osModule.vmModule.blockManagerType = "NullBlockManager"
TestCloud.vm_medium[*].osModule.vmModule.blockManager.numBlockServers = 1
TestCloud.vm_medium[*].osModule.vmModule.schedulerType = "NullBlockScheduler"
TestCloud.vm_medium[*].osModule.vmModule.cacheType = "NullCache"
### File_System [0]:NFS_FS
```

```
TestCloud.vm_medium[*].osModule.fsModule[0].fsType = "Basic_FileSystem"
TestCloud.vm_medium[*].osModule.fsModule[0].fs.preLoadFiles = true
# Application [1] : DummyApplication
TestCloud.vm_medium[*].appModule[*].appType = "DummyApplication"
TestCloud.vm_medium[*].appModule[*].app.application_netType = "INET"

### Main Node parameters
TestCloud.vm_medium[*].numCPUs = 2
TestCloud.vm_medium[*].numBlockServers = 1
TestCloud.vm_medium[*].numFS = 1
TestCloud.vm_medium[*].numApps = 10
TestCloud.vm_medium[*].hostName = "medium[*]"

######################################################################
###     Definition of switch DefaultSwitch_0
######################################################################
TestCloud.switch_medium.relayUnit.numCPUs = 2
TestCloud.switch_medium.relayUnit.processingTime = 5us
TestCloud.switch_medium.relayUnit.bufferSize = 1MB

######################################################################
###     Definition of node vm_high[*]
######################################################################
### CPU system
TestCloud.vm_high[*].cpuModule.cpuCoreType = "CPUcore"
TestCloud.vm_high[*].cpuModule.CPUcore[*].speed = 5
TestCloud.vm_high[*].cpuModule.CPUcore[*].tick_s = 0.001
### Storage system
TestCloud.vm_high[*].bsModule[*].diskType = "Disk_400GB_LI"
TestCloud.vm_high[*].bsModule[*].cacheType = "NullCache"
### Operating System
TestCloud.vm_high[*].osModule.networkServiceType = "NetworkService"
TestCloud.vm_high[*].osModule.cpuSchedulerType = "CPU_Scheduler_FIFO"
TestCloud.vm_high[*].osModule.memoryType = "BasicMainMemory"
### CPU Scheduler
TestCloud.vm_high[*].osModule.cpuScheduler.numCPUs = 4
### Memory
TestCloud.vm_high[*].osModule.memory.readLatencyTime_s = 1
TestCloud.vm_high[*].osModule.memory.writeLatencyTime_s = 1
TestCloud.vm_high[*].osModule.memory.size_KB = 4000000
TestCloud.vm_high[*].osModule.memory.blockSize_KB = 128
```

```
### Volume Manager
TestCloud.vm_high[*].osModule.vmModule.blockManagerType = "NullBlockManager"
TestCloud.vm_high[*].osModule.vmModule.blockManager.numBlockServers = 1
TestCloud.vm_high[*].osModule.vmModule.schedulerType = "NullBlockScheduler"
TestCloud.vm_high[*].osModule.vmModule.cacheType = "NullCache"
### File_System [0]:NFS_FS
TestCloud.vm_high[*].osModule.fsModule[0].fsType = "Basic_FileSystem"
TestCloud.vm_high[*].osModule.fsModule[0].fs.preLoadFiles = true
# Application [1] : DummyApplication
TestCloud.vm_high[*].appModule[*].appType = "DummyApplication"
TestCloud.vm_high[*].appModule[*].app.application_netType = "INET"
### Main Node parameters
TestCloud.vm_high[*].numCPUs = 4
TestCloud.vm_high[*].numBlockServers = 1
TestCloud.vm_high[*].numFS = 1
TestCloud.vm_high[*].numApps = 10
TestCloud.vm_high[*].hostName = "high[*]"

##########################################################################
###     Definition of switch DefaultSwitch_0
##########################################################################
TestCloud.switch_high.relayUnit.numCPUs = 4
TestCloud.switch_high.relayUnit.processingTime = 5us
TestCloud.switch_high.relayUnit.bufferSize = 1MB
### Configuration files
TestCloud.fileConfig.iorConfigFile = "ior.txt"
TestCloud.fileConfig.serversNfs = ""
TestCloud.fileConfig.serversPfs = ""
TestCloud.fileConfig.preloadFiles = "preLoadFiles.txt"
TestCloud.fileConfig.mpiGraphEnv = ""
TestCloud.fileConfig.mpiEnv = ""

include netFeatures.ini
```

Figura 57: Fichero de configuración omnetpp.ini

La anterior Figura 57 muestra un ejemplo de fichero omnetpp.ini. En él se recoge la definición de las máquinas virtuales con las que cuenta el cloud a simular, describiendo todos y cada uno de sus componentes (memoria, CPU,…) así como los switch que comunican una máquinas con otras.

scenario.ned

```
package iCanCloud.simulations.TestCloud;
import ned.DatarateChannel;
import inet.nodes.ethernet.EtherSwitch2;
import iCanCloud.VirtualMachines.VMs.*;
import iCanCloud.VirtualMachines.Racks.*;
import iCanCloud.Hypervisor.Hypervisor;
import inet.networklayer.autorouting.ParallelNetworkConfigurator;
import iCanCloud.Base.FileConfigManager;

// ------------------------------------------------------------
//     Definition of channels
// ------------------------------------------------------------
channel vmChannel extends DatarateChannel
{
    delay = 250.0us;
    datarate = 1Gbps;
}

network TestCloud
{
    parameters:
        int numVM_medium;

        int numVM_high;

    // ------------------------------------------------------------
    //     Definition of main modules
    // ------------------------------------------------------------
    submodules:

        // ------------------------------------------------------------
        //     Network configurator (parallel simulation)
        // ------------------------------------------------------------
```

```
        netConfig: ParallelNetworkConfigurator {
            parameters:
                ipFile = "ips.txt";
                networkAddress = "192.168.0.0";
                netmask = "255.255.255.0";
        }
        fileConfig: FileConfigManager {
            parameters:
                @display("i=iCanCloud/file_config/filecfg;p=50,50");
}
// -------------------------------------------------------------
//     Definition of virtual machines
// -------------------------------------------------------------
        vm_medium[numVM_medium]: VM_TCP {
        @display("p=58,156");
       }
       vm_high[numVM_high]: VM_TCP {
        @display("p=58,156");
       }
        // -------------------------------------------------------------
        //     Definition of switches
        // -------------------------------------------------------------
        switch_medium: EtherSwitch2 {
            gates:
                ethg[numVM_medium];
        }
        switch_high: EtherSwitch2 {
            gates:
                ethg[numVM_high];
        }
        hypervisor: Hypervisor {
            parameters:
                jobListFile = "C:\Documents and Settings\pani\Mis
documentos\iCanCloud\simulations\TestCloud\TestCloud.cfg";
        }
    // -------------------------------------------------------------
    //     Connection between modules
    // -------------------------------------------------------------
    connections allowunconnected:


        // -------------------------------------------------------------
```

```
        //     Connection between DefaultSwitch_medium and DefaultNode_medium
        // ---------------------------------------------------------------
        for i=0..numVM_medium-1 {
            switch_medium.ethg++ <--> vmChannel <--> vm_medium[i].ethg++;
        }
        // ---------------------------------------------------------------
        //     Connection between DefaultSwitch_high and DefaultNode_high
        // ---------------------------------------------------------------
        for i=0..numVM_high-1 {
            switch_high.ethg++ <--> vmChannel <--> vm_high[i].ethg++;
        }
}
```

Figura 58: Fichero de configuración scenario.ned

En la Figura 58 se muestra un ejemplo de fichero scenario.ned como el que deberá generar el sistema a desarrollar. Este archivo describe cómo es la red en la que se ejecutarán las aplicaciones definidas (cómo es la estructura del cloud) a partir de la información almacenada en otros ficheros como el omnetpp.ini o el fichero .cfg.

ips.txt

```
TestCloud.vm_medium[0]:192:168:0:1
TestCloud.vm_medium[1]:192:168:0:2
TestCloud.vm_medium[2]:192:168:0:3
TestCloud.vm_medium[3]:192:168:0:4
TestCloud.vm_medium[4]:192:168:0:5
TestCloud.vm_medium[5]:192:168:0:6
TestCloud.vm_medium[6]:192:168:0:7
TestCloud.vm_medium[7]:192:168:0:8
TestCloud.vm_medium[8]:192:168:0:9
TestCloud.vm_medium[9]:192:168:0:10
TestCloud.vm_medium[10]:192:168:0:11
TestCloud.vm_medium[11]:192:168:0:12
TestCloud.vm_medium[12]:192:168:0:13
TestCloud.vm_medium[13]:192:168:0:14
TestCloud.vm_medium[14]:192:168:0:15
TestCloud.vm_medium[15]:192:168:0:16
…..
```

Figura 59: Fichero de configuración ips.txt

La Figura 59 anterior muestra el último de los ficheros que deberá generar nuestro sistema, el ips.txt. En este archivo se asocia una dirección IP a cada una de las máquinas virtuales que compondrán el cloud, con objeto de definir las comunicaciones entre las mismas.

4.8 Plataforma de desarrollo

Si bien en apartados anteriores hacíamos hincapié en cómo podía dividirse el sistema a desarrollar desde el punto de vista de la funcionalidad ofrecida, nos encontramos ahora con otro enfoque a la hora de dividir la aplicación de forma modular: este enfoque corresponde a las vías de comunicación con las que cuenta la herramienta.

Como ya sabemos, la aplicación debe comunicarse con el cliente para la recogida o entrega de la información pertinente. Pero la aplicación tiene otro agente muy importante con el que debe interactuar, este es, el propio simulador.

En la Figura 60 se muestran las dos vías de comunicación con las que cuenta la aplicación, como se comentó en párrafos anteriores:

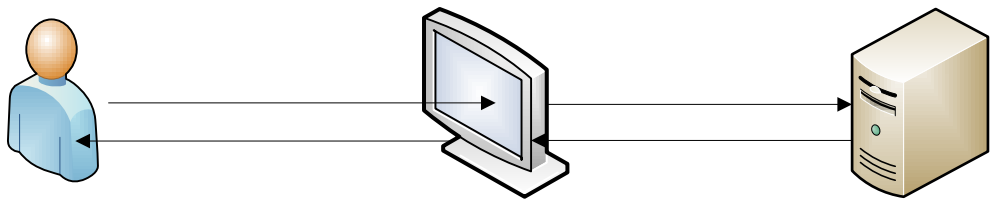

Figura 60: Comunicación usuario-aplicación-simulador

El simulador "iCanCloud" ha sido implementado utilizando el lenguaje de programación C y funciona bajo cualquier máquina con un sistema operativo basado en Linux (Ubuntu, Debian, etc.). Por ello, es requisito imprescindible que la aplicación a desarrollar sea capaz de comunicarse con cualquier sistema basado en Linux para poder asegurar esta interacción con el simulador.

Por otro lado, la aplicación debe poder comunicarse con el usuario de la misma mediante una interfaz debidamente diseñada. Ahora bien, ¿cuál es el perfil generalizado de este usuario? El público al que va destinada la herramienta a desarrollar se localiza principalmente en el sector comercial y de servicios, es decir, empresas que quieren entrar en el campo de la computación en nube para ofrecer o contratar una serie de servicios. Las empresas pertenecientes a este sector generalmente optan por soluciones software para sus equipos basadas en sistemas

operativos de la familia de Microsoft, como pueda ser Microsoft Windows XP. De hecho, este sistema es, incluso hoy en día, el sistema operativo más extendido en este ámbito.

Pero esta herramienta también está destinada a grupos de investigación de distintas áreas dentro del cloud computing. En este nuevo colectivo prolifera la tendencia de trabajar con sistemas operativos basados en Linux, además de los mencionados basados en productos de Microsoft.

Nos encontramos entonces ante la necesidad de que la herramienta funcione indistintamente tanto en sistemas basados en Linux como en Windows. Por ello, la elección del lenguaje de programación utilizado para la implementación de la aplicación estará condicionada por la característica de multiplataforma del propio lenguaje.

De entre todos los lenguajes multiplataforma que se conocen hoy en día se ha elegido el **lenguaje de programación Java** como el ideal para este propósito. Si bien existe otros lenguajes muy populares como C#, estos no aportan una característica de multiplataforma tan contrastada como la que proporciona Java.

Además, este lenguaje orientado a objetos se caracteriza por ser un lenguaje robusto y seguro y estar ampliamente extendido en distintos entornos de funcionamiento como dispositivos móviles, navegadores web, servidores, etc, por lo que podría permitir distintas posibilidades de ampliación en un futuro.

Capítulo 5

5. Caso práctico

En este apartado se va a mostrar un ejemplo práctico de configuración de un cloud y posterior simulación del mismo. Si bien, como ya se comentó en apartados anteriores, el sistema es multiplataforma, se ha decidido utilizar la versión Windows para la realización de este ejemplo por simplicidad.

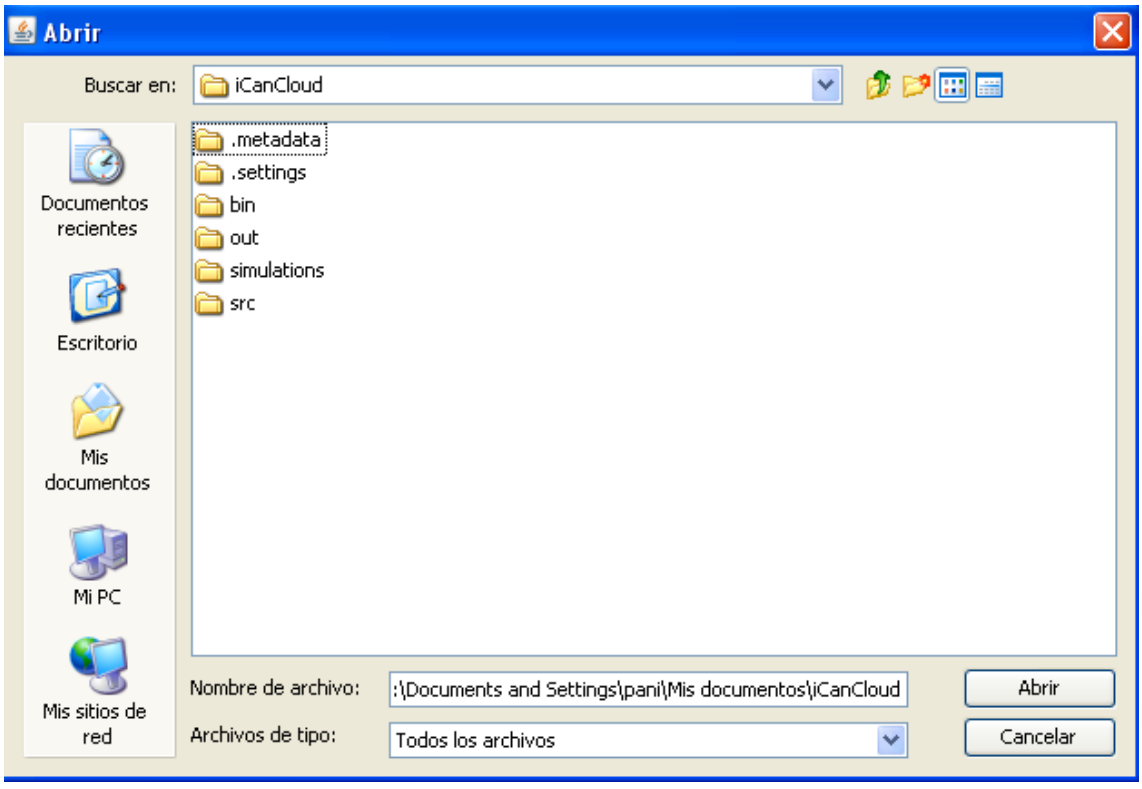

Figura 61: Pantalla de selección del espacio de trabajo de iCanCloud

Al ejecutar la aplicación, el sistema nos pedirá que seleccionemos el espacio de trabajo del simulador iCanCloud. En la Figura 61 se muestra la ventana emergente en la que el usuario de la aplicación debe seleccionar la carpeta base del simulador. Gracias a este enlace, la herramienta tendrá acceso a los datos del simulador para, por ejemplo, cargar dinámicamente los tipos de aplicaciones incluidos en el simulador, generar adecuadamente los ficheros de configuración o lanzar la propia simulación.

Una vez que se ha seleccionado adecuadamente el espacio de trabajo del simulador se mostrará la pantalla de bienvenida de la aplicación. En la Figura 62 se puede ver cómo al arrancar la misma se cargan todos los datos de configuración que el usuario hubiera almacenado durante la última sesión.

Figura 62: Pantalla de bienvenida de la aplicación

Así, en la parte izquierda de la interfaz aparecen dos árboles de nodos: el de arriba es el árbol de elementos del repositorio mientras que el de abajo es el árbol de simulaciones. Al arrancar la aplicación se carga en el árbol superior todos los elementos del sistema, es decir, máquinas virtuales, aplicaciones, tareas de usuario y clouds. El sistema lee todos los elementos almacenados en el repositorio como ficheros XML y los carga en la interfaz de forma que el

usuario pueda tener acceso a ellos tal y como estaban configurados cuando cerró la última sesión en el sistema.

En primer lugar, vamos a crear una nueva máquina virtual en el sistema, la cual vamos a denominar con el nombre de “small”. Para crear una nueva MV pinchamos con el botón derecho sobre el nodo “VirtualMachines” en el árbol de elementos y elegimos la opción “Add new virtual machine”. En el panel que se abre introducimos los valores que creamos oportunos y pulsamos sobre el botón “Apply”.

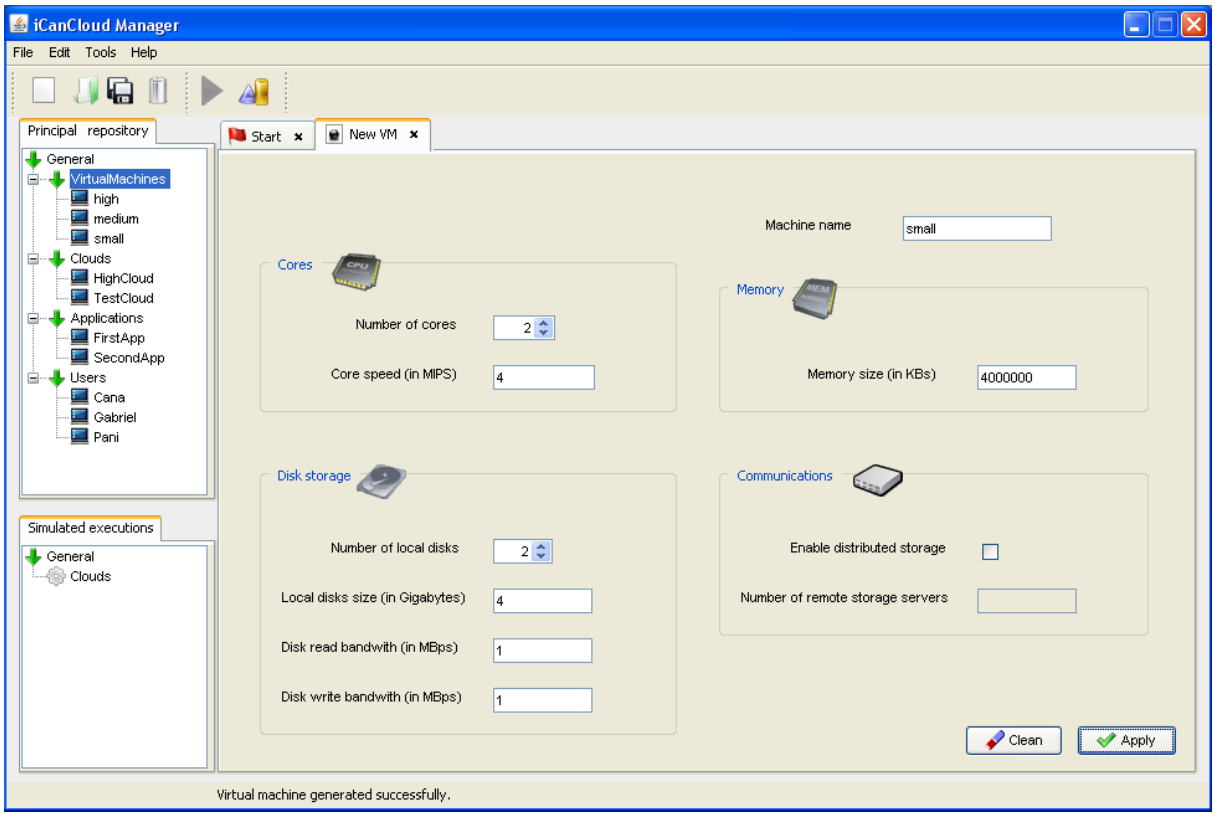

Figura 63: Pantalla de creación de una máquina virtual

En la Figura 63, se muestra el resultado del proceso anterior. Si se han seguido los pasos adecuadamente y se han rellenado todos los parámetros (número de núcleos, velocidad de cada núcleo en MIPS) se generará la nueva máquina virtual en el sistema. Aparecerá así un nuevo nodo en el árbol superior izquierdo con el identificador de la nueva máquina creada.

A continuación vamos a crear una nueva aplicación que se ejecute en un cloud del sistema. Para ello, de manera similar, seleccionamos el nodo Applications del subárbol superior izquierdo y haciendo click derecho elegimos la opción “Add new application”. Seleccionamos que la nueva aplicación sea del tipo que queramos e introducimos los valores oportunos para

sus parámetros. Para finalizar, pulsamos sobre el botón Apply y si todo es correcto (la aplicación no ha sido previamente definida en el sistema) se añade su identificador al árbol del repositorio. En la Figura 64 que aparece a continuación se muestra un ejemplo de este procedimiento.

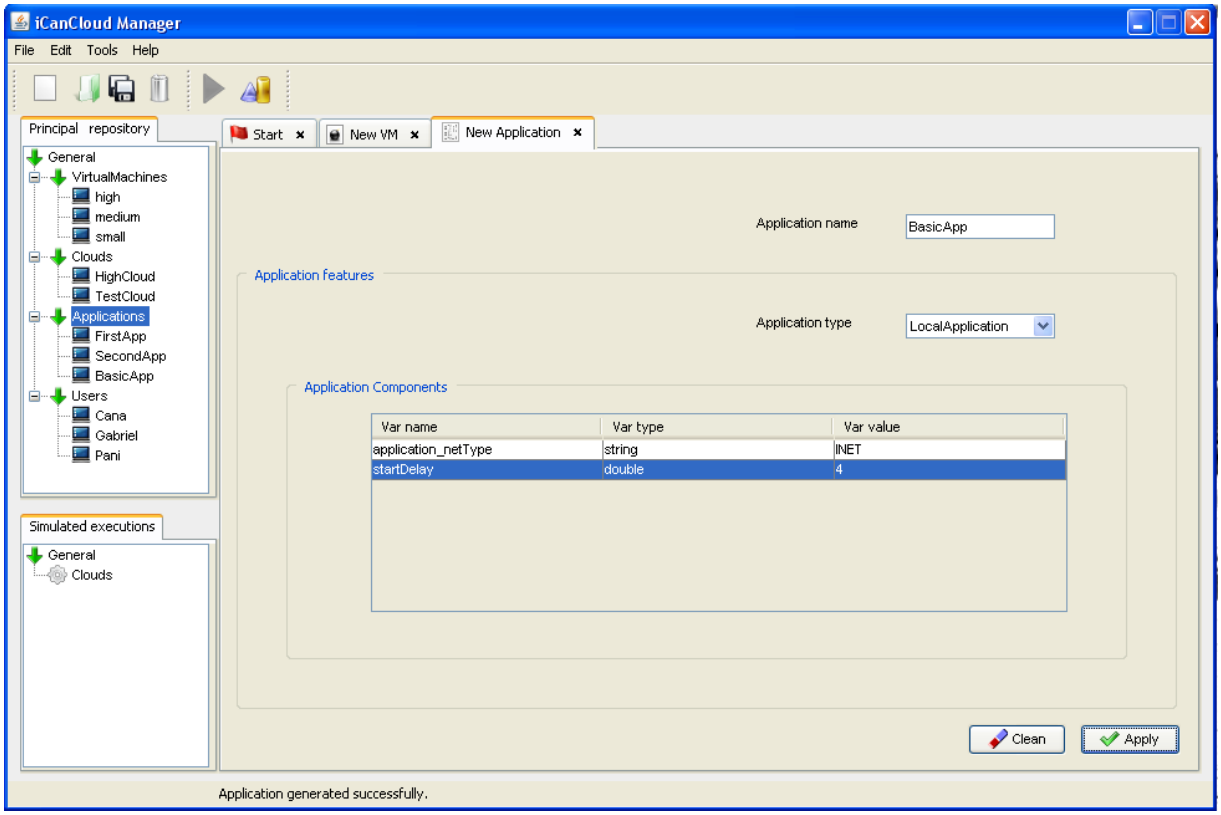

Figura 64: Pantalla de creación de una nueva aplicación

A continuación, vamos a modificar una tarea de usuario existente para hacer que incluya la aplicación que acabamos de generar. Imaginemos que el usuario “Cana” quiere probar esta aplicación en el cloud. Así, se selecciona el nodo correspondiente en el subárbol Users (tareas del usuario), se hace click derecho y se elige la opción “Edit User”.

De esta forma, aparece un nuevo panel correspondiente a los trabajos elegidos por el usuario anterior para ejecutar sobre el cloud. Esta tarea tiene en el momento dos trabajos asociados: la ejecución de una aplicación FirstApp sobre tres máquinas virtuales de tipo high con dos iteraciones y la ejecución de una aplicación SecondApp sobre otras tres máquinas high, esta vez en una sola iteración.

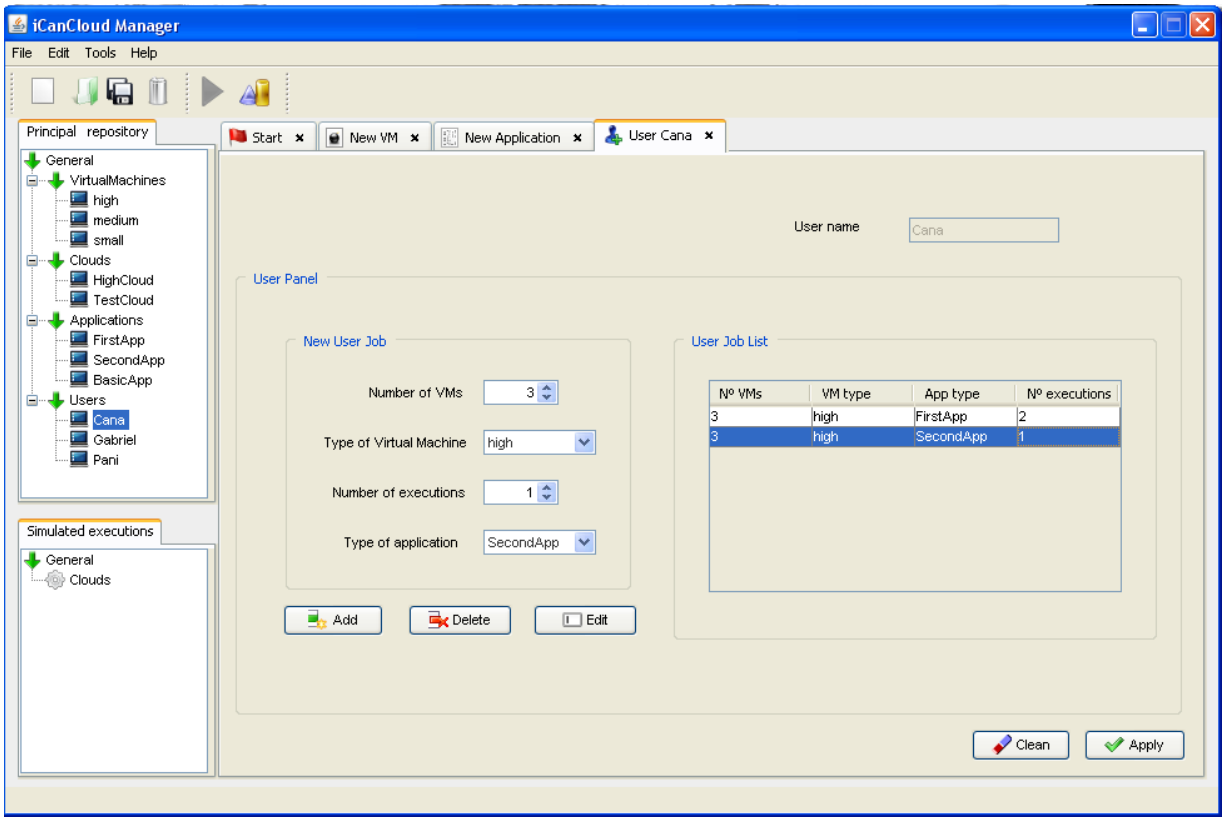

Figura 65: Pantalla de modificación de la tarea de usuario

Vamos a modificar estos trabajos para que en lugar de las aplicaciones que se definen se ejecute la aplicación que acabamos de generar. Para ello, seleccionamos cada una de las entradas de la tarea y elegimos la aplicación BasicApp. En uno de los casos elegiremos que se ejecute sobre máquinas de tipo high y en el otro caso se ejecutará sobre las nuevas máquinas virtuales “small” que acabamos de crear.

Cuando hemos terminado de modificar la tarea le damos al botón Apply y la interfaz nos devolverá un mensaje diciendo que hemos modificado correctamente la tarea.

En la Figura 66 se muestra cómo la tarea de usuario ha sido modificada para incluir los nuevos elementos añadidos al sistema.

Para finalizar el proceso de configuración vamos a modificar uno de los clouds existentes en el sistema para incluir la tarea de usuario que ha sido modificada y poder obtener una serie de resultados. De esta forma seleccionaremos el cloud “TestCloud” del árbol de repositorios y elegiremos la opción “Edit Cloud”.

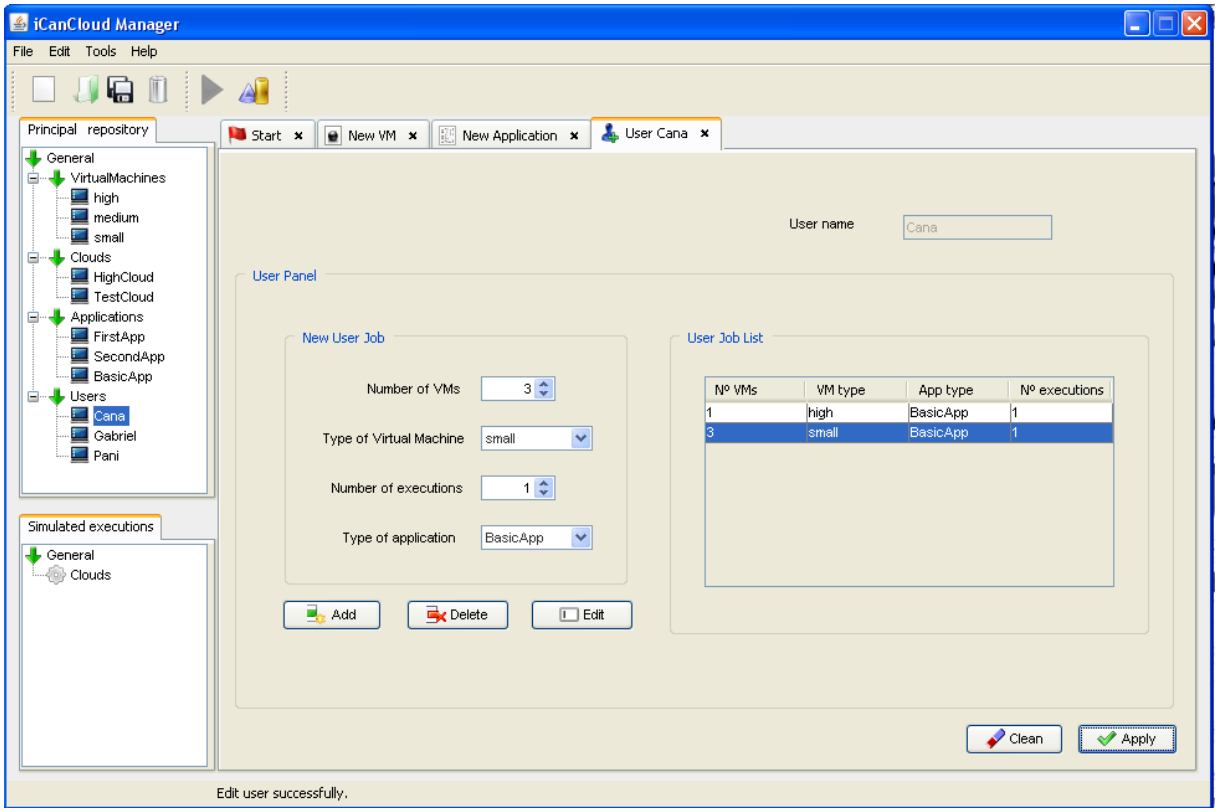

Figura 66: Pantalla de tarea de usuario después de la modificación

El panel de configuración del cloud contiene dos secciones principales, tal y como se ha descrito a lo largo de este documento. Por un lado, en la parte superior, contiene la infraestructura ofrecida por el proveedor, es decir, las máquinas virtuales de las que se dispone y el coste de cada una de ellas. Por su parte, en la parte inferior se reflejan las tareas de usuario que se desean ejecutar en el cloud, cada una de ellas con las máquinas que se necesitan (como vimos cuando creamos la tarea anterior).

Como puede apreciarse en la Figura 67, el cloud ya contiene la tarea de usuario “Cana” entre la lista de tareas de debe ejecutar (tabla inferior). Por ello, en principio no sería necesario modificarla para incluir dicha tarea. Sin embargo, la ejecución de este cloud tal y como está configurado no funcionará. Si recordamos los requisitos del sistema, se comentó que una tarea de usuario no puede solicitar la ejecución de una serie de aplicaciones sobre máquinas virtuales que no estén disponibles en el sistema (que no hayan sido definidas en la parte superior de este panel). Así, no sería posible ejecutar las aplicaciones BasicApp sobre las máquinas virtuales de tipo “small”, tal y como se definió en la tarea.

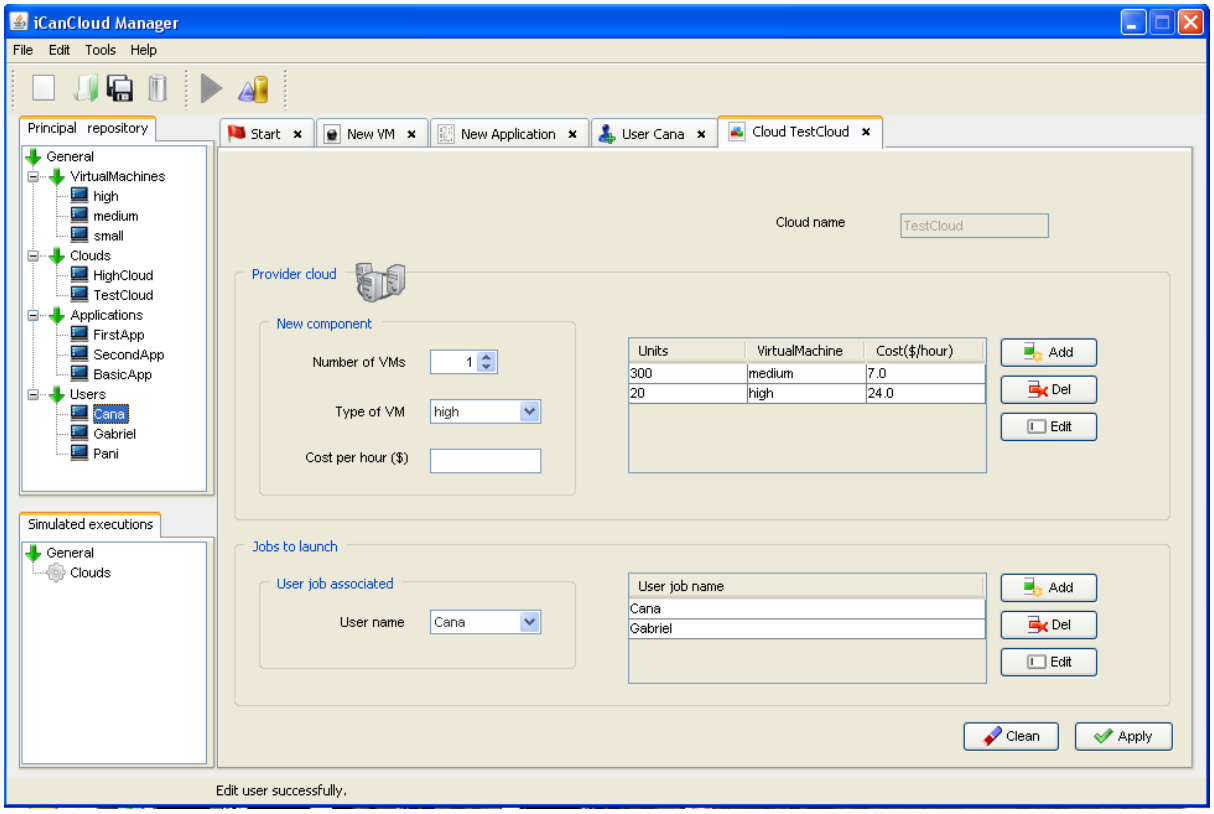

Figura 67: Pantalla de modificación de cloud

Para solucionar este problema, es necesario añadir una nueva línea en la tabla superior (infraestructura del cloud) donde se especifique que se dispone de máquinas “small” en el cloud. Pero además el número de máquinas de este tipo de las que se dispone debe ser mayor al número de máquinas que se solicitan en la tarea.

Después de añadir dichas máquinas a la infraestructura del cloud, aplicamos los cambios y si se cumplen todas estas restricciones en el cloud, se mostrará un nuevo mensaje en la interfaz confirmando la operación.

En la Figura 68 se muestra un ejemplo de este proceso de modificación del cloud y su confirmación en la interfaz.

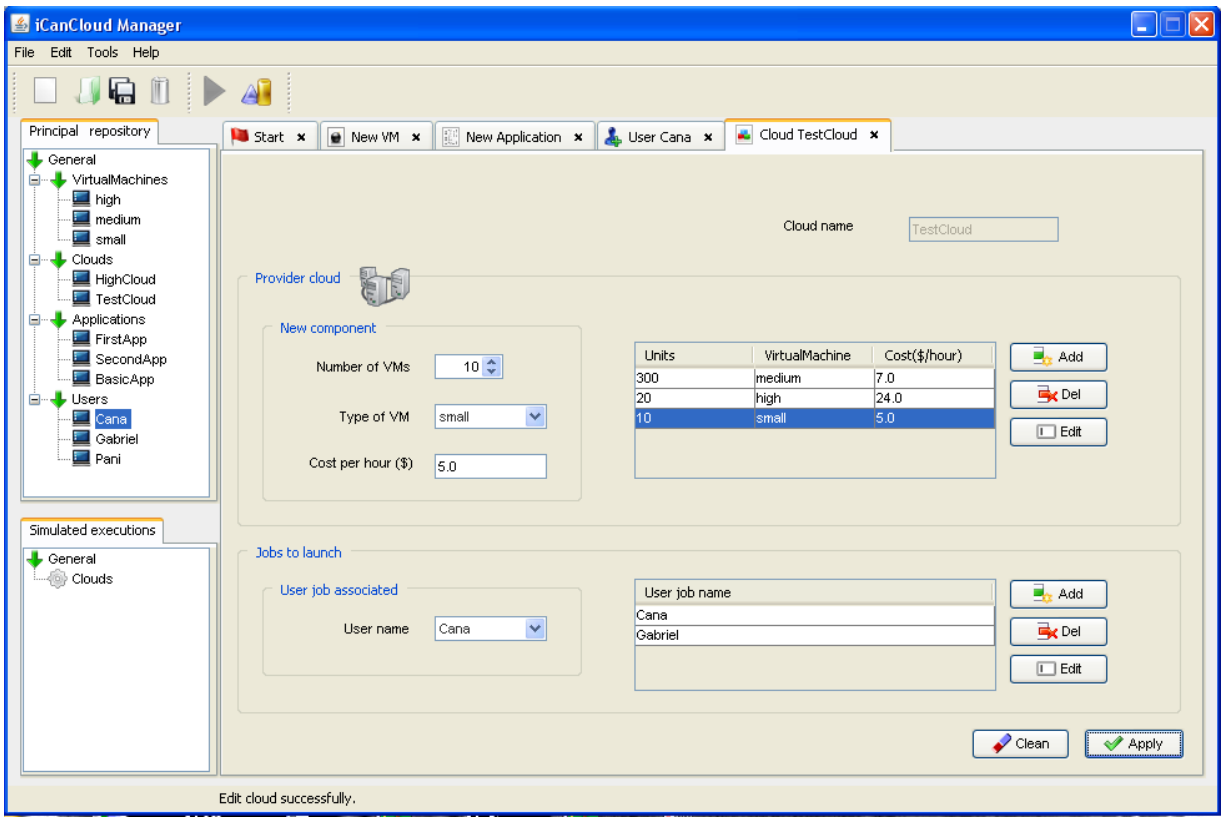

Figura 68: Pantalla de cloud tras la modificación

Una vez que se ha configurado adecuadamente el cloud solo queda lanzar la simulación. Para ello, seleccionando el cloud que queramos, apretamos el botón Run de la barra de herramientas superior (símbolo de play). De esta forma, se abrirá un nuevo panel en el que se mostrará al usuario la salida por pantalla del simulador.

En la Figura 69 se muestra una captura de pantalla de este proceso. Después de la ejecución se añade un nuevo nodo en el árbol de simulaciones (en la parte inferior izquierda de la pantalla). Los datos que aparecen en pantalla son información valiosa de cara al programador pero no sirven para el cliente de la aplicación, el cual es ajeno a este tipo de datos.

Sin embargo, el usuario puede hacer click derecho sobre el nodo creado en el árbol de simulaciones y elegir la opción “View graphic results”, con lo que se cargarán las gráficas generadas por el sistema que utilizan los datos obtenidos por la simulación realizada.

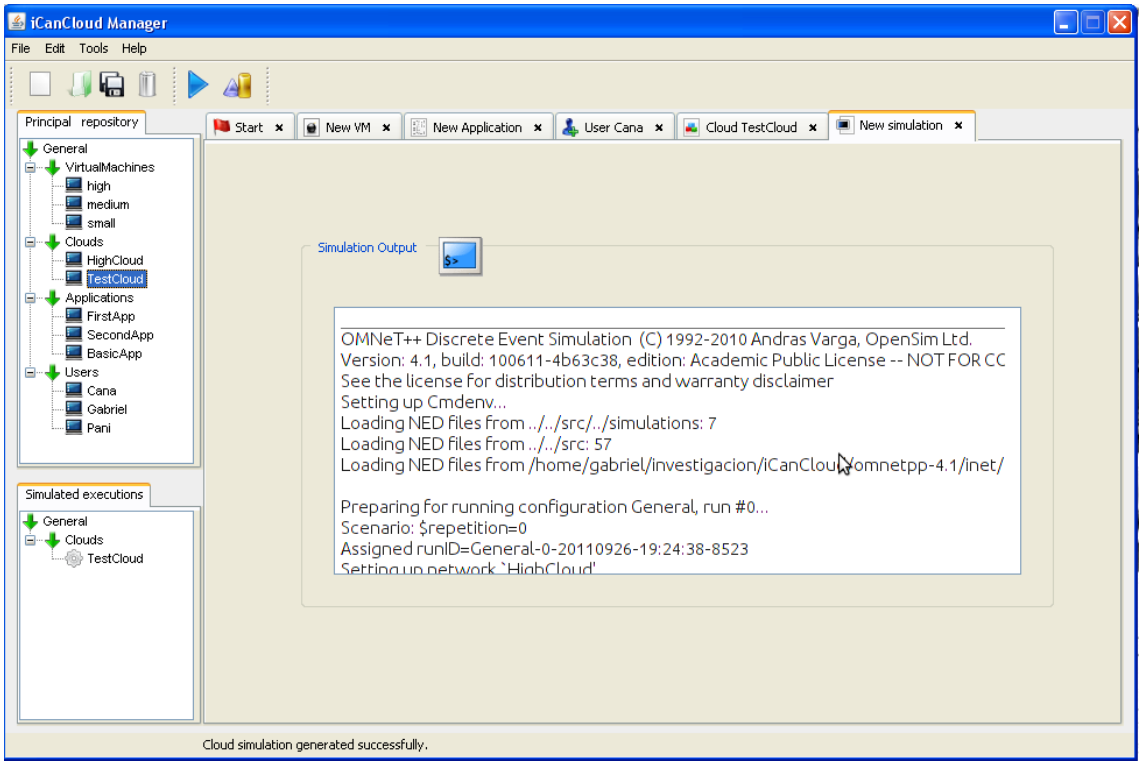

Figura 69: Pantalla de simulación del cloud

En la Figura 70 se muestra la pantalla de resultados de la simulación tal y como se comentó anteriormente. En esta pantalla aparecen varios elementos que se describen a continuación.

En la parte inferior de la misma, se muestra un carrusel de imágenes, donde se cargan todas las gráficas que han sido generadas como resultado de la ejecución de la simulación en el sistema. Presionando las flechas izquierda y derecha se puede navegar por todas las imágenes generadas.

En la parte superior aparece una gráfica de resultados. Esta imagen corresponde a la imagen que tenemos seleccionada del carrusel inferior. Así, cada vez que pinchamos sobre una gráfica distinta, ésta se carga en el panel superior para poder estudiar sus resultados de forma óptima.

En la parte derecha aparece un cuadro que muestra la información del elemento que estamos visualizando en el panel superior. Cada gráfica hace referencia a los resultados obtenidos de la ejecución de una tarea concreta en una máquina virtual determinada. Así, este panel muestra información sobre la tarea de usuario que contiene esa aplicación, la aplicación en cuestión que se ha ejecutado, el tipo de aplicación, el número de máquinas virtuales que se han utilizado, etc.

Con estos datos y el resultado de las gráficas, el cliente del sistema tiene una visión más global de los resultados obtenidos por la simulación realizada.

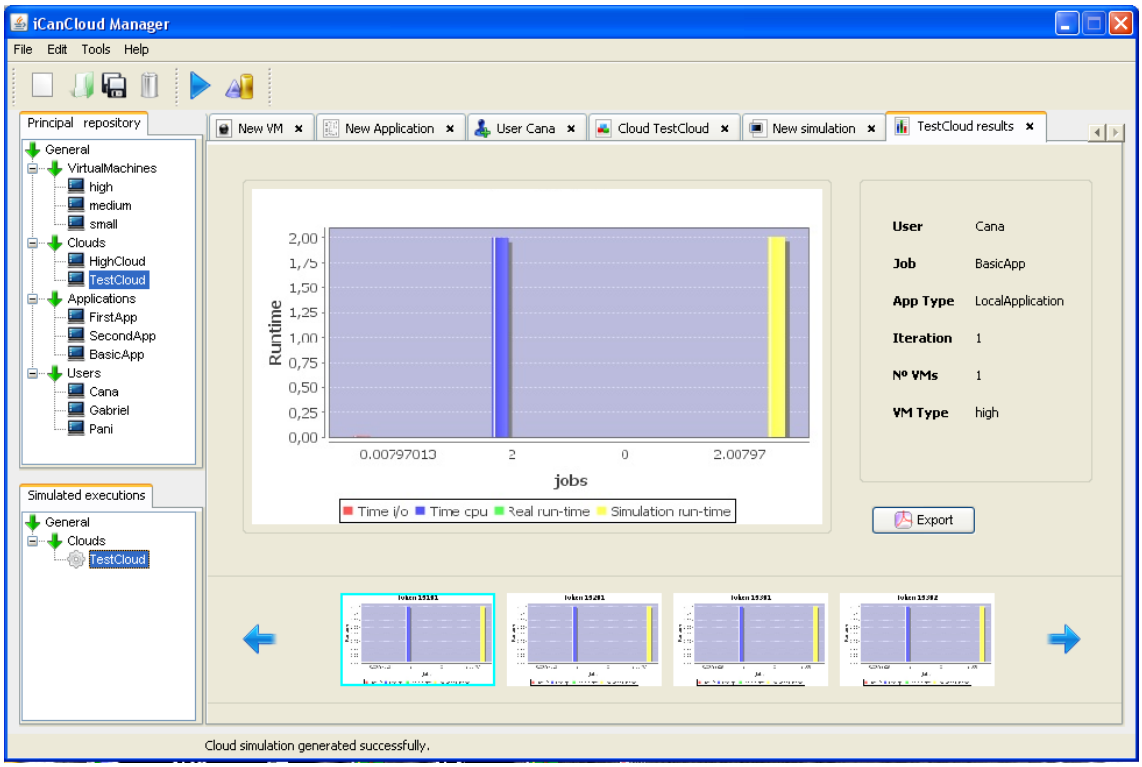

Figura 70: Pantalla de resultados gráficos de la simulación

Por último, debajo de este panel de datos hay un botón que permite exportar todos los datos de la simulación a un informe en formato PDF con una estructura determinada. Se genera de esta forma un fichero en el que se almacenan todos los datos del cloud simulado, así como los resultados obtenidos y las gráficas que se han generado a partir de ellos.

En la Figura 71 se muestra un ejemplo de este proceso de generación del informe en formato PDF.

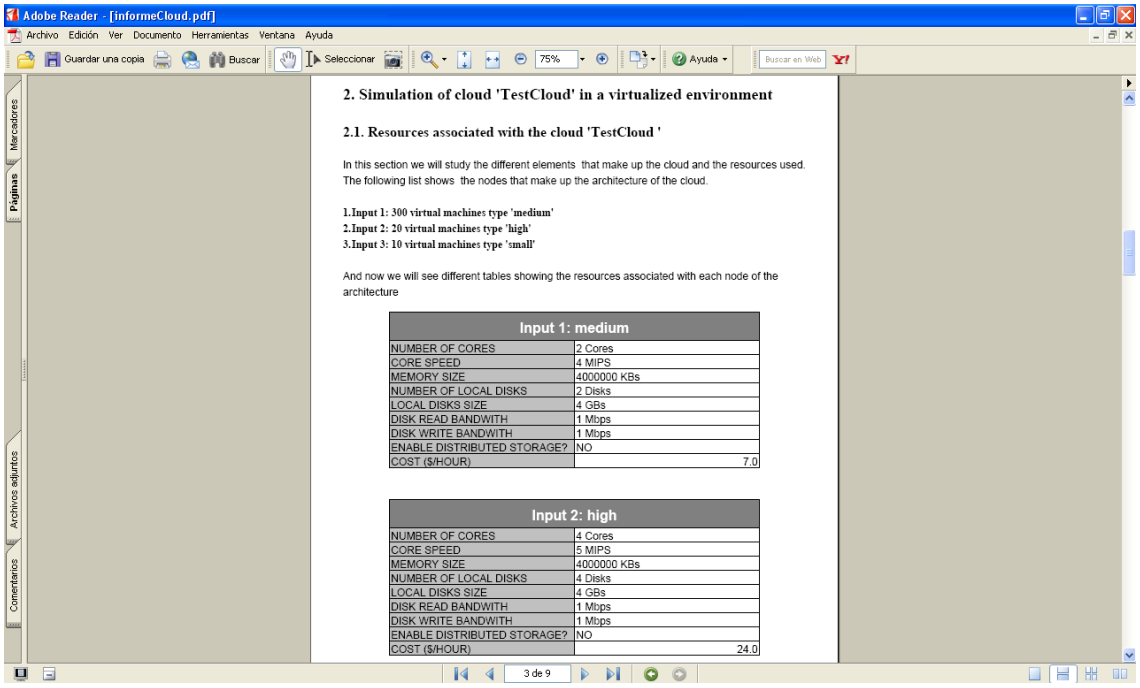

Figura 71: Pantalla de informe de resultados

Capítulo

6

6. Conclusiones y líneas futuras

Llegados a este punto, en el que ha finalizado el desarrollo del proyecto, y se han obtenido y estudiado los resultados finales, es posible llegar a una serie de conclusiones.

Por un lado, la elección del presente trabajo como Proyecto Final de Carrera me ha parecido muy acertada. El estudio de enfoques de experimentación alternativos en el campo del cloud computing, como son las simulaciones, es un importante tema de debate actual y se convertirá a buen seguro en una de las principales líneas de investigación en un futuro próximo. Se trata de una práctica cada día más utilizada por los clientes, ya que es posible reducir los costes asociados a las pruebas y evaluar diferentes tipos de escenarios de recursos de baja carga y distribución de precios variables.

En lo que a las fases previas a la implementación se refiere, se puede comentar que la elección del modelo utilizado en el diseño de la arquitectura (Modelo Vista Controlador) ha sido la adecuada dada la naturaleza del proyecto. Este modelo se adapta perfectamente a aquellos proyectos en los que la "Vista" tiene un papel importante y se desea aislarla del modelo. Así, es posible tener varias vistas distintas manteniendo un modelo de datos común. En nuestro caso, el modelo se ha diseñado e implementado siguiendo una línea genérica, de modo que es muy difícil que cambie; sin embargo, la vista puede ser adaptada dependiendo del entorno en que se vaya a trabajar con la aplicación (Windows, Unix,…).

En relación a esta característica tiene mucho que decir también el lenguaje de programación escogido, Java 2EE. Al tratarse de un lenguaje multiplataforma, permite la ejecución del sistema en cualquier sistema que incorpore un entorno de ejecución de Java o JRE (Java Runtime Environment), por lo que su capacidad de distribución y escalabilidad aumenta considerablemente.

En lo que respecta a la comunicación de la herramienta con el simulador *iCanCloud*, es necesario comentar que no ha resultado una tarea tan trivial como se pensó en un principio.

Ha sido necesario realizar una tarea de adaptación entre ambos sistemas para permitir su comunicación (generación de resultados por parte del simulador que fueran legibles a la herramienta, adaptación de los ficheros de configuración,…).

Otro de los puntos que ha resultado más problemático y difícil de resolver es el de la carga dinámica de aplicaciones. Como se ha comentado durante todo el documento, la idea inicial era desarrollar un sistema lo más genérico posible que permitiera al cliente simular cualquier tipo de trabajo. Cada trabajo o tarea, como ya se vio, se basa en un conjunto de aplicaciones a ejecutar sobre el cloud. Para que un cliente pueda ejecutar una aplicación propia, debe crearse un modelo matemático que represente su comportamiento y seguir un proceso complicado de adaptación. Desde el punto de vista de la herramienta, la gestión de aplicaciones de este tipo, donde no se conoce su estructura, resulta una tarea muy complicada de gestionar.

Es fácil adivinar cómo además, uno de los principales aspectos que se han mejorado con la utilización de la herramienta conjuntamente con el simulador *iCanCloud*, es el tiempo empleado en la puesta a punto de la simulación. Como pudimos ver en el apartado 4.7 Sintaxis de los ficheros de configuración, son varios los ficheros que necesita el simulador para lanzar cada ejecución. Una de las tareas principales de la herramienta es generar estos ficheros de configuración de manera automática a partir de la información asociada al cloud y lanzar la ejecución con dichos parámetros. Por lo tanto este proceso se ha agilizado enormemente y el tiempo empleado se ha reducido drásticamente.

A partir de la herramienta actual, podrían realizar una serie de mejoras en un futuro, las cuales podrían ir enfocadas según los siguientes puntos:

- Permitir la comunicación con el simulador iCanCloud, sin tener la necesidad de que la herramienta resida en el mismo equipo que el propio simulador. Podría facilitarse la comunicación por otros medios, como por ejemplo por SSH, de forma que el sistema se comunicara con otra máquina remota, donde residiría el simulador. De esta forma, aumentaría las posibilidades de distribución de la aplicación.

- Permitir en un futuro el estudio de otro tipo de resultados y no solo tiempos y costes tal y como sucede con la versión actual de la herramienta. Se podría, por ejemplo, medir la congestión de la red en un cloud en base al tráfico que circula por ella, etc.

- Dado que el simulador iCanCloud es altamente escalable, es posible que en un futuro se permitiera la comunicación entre clouds, por lo que la herramienta de gestión de entornos debería estar preparada para contemplar este tipo de casos.

Anexo 1

7. Anexo I – Documento de costes

En este documento anexo se va a realizar una planificación de la producción del software, labor para la cual se ha decidido optar por una solución basada en el modelo de costes conocido por el nombre de **COCOMO**.

7.1 Estimación de costes

El Modelo Constructivo de Costes COCOMO (del inglés, Constructive Cost Model), es un modelo matemático de base empírica utilizado para la estimación de costes de software. Incluye tres submodelos, cada uno con un nivel de detalle cada vez mayor, a medida que avanza el proceso de desarrollo del software: estos son, el modelo básico, el modelo intermedio y el modelo detallado.

Para realizar una estimación de coste del proyecto basándonos en esta metodología, es necesario tener en cuenta diversos factores, como son el modelo de estimación, los factores de escala, los multiplicadores de esfuerzo y los puntos de función.

En lo que respecta al modelo de estimación, COCOMO contempla dos tipos distintos: el modelo pre-arquitectura y el modelo post-arquitectura. El primero de ellos explora las arquitecturas software alternativas y conceptos de operación, mientras que el segundo incluye el desarrollo y el mantenimiento de la aplicación. Es en éste último modelo (post-arquitectura) en el que se utilizan puntos de función con modificadores para la reutilización y objetos de software y, puesto que ya se conoce toda la información requerida, será éste el modelo elegido.

Otro punto a tener en cuenta es el relativo a los factores de escala, los cuales afectan de forma global a todo el proyecto en lo que se refiere a flexibilidad, cohesión del equipo de desarrollo, experiencia previa, etc.

Los multiplicadores de esfuerzo son otros elementos a contemplar en la planificación. Así el esfuerzo nominal del desarrollo se ajusta para una mejor estimación mediante factores que se clasifican en producto, plataforma, personal y proyecto.

Finalmente, se miden los puntos de función, los cuales son la base para la medición del tamaño de los modelos de estimación pre y post-arquitectura. Estos, tienen como objetivo cuantificar la funcionalidad de un sistema. Son estimadores muy útiles puesto que están basados en información disponible en etapas tempranas del ciclo de vida del desarrollo software.

7.1.1 Modelo de estimación

Como se comentó en el apartado anterior, el modelo de estimación escogido es el de post-arquitectura, ya que se conoce hasta el momento toda la información requerida para su utilización. Este hecho facilitará que se consiga una planificación más precisa del esfuerzo y el coste del proyecto.

Los factores que se han tenido en cuenta para la elección del modelo de estimación post-arquitectura son los siguientes:

- **Tamaño del proyecto**: se estimará a continuación mediante la utilización de la metodología de los puntos función.
- **Plataforma a utilizar**: como se especificó en el apartado correspondiente al Diseño del sistema, la aplicación se desarrollará sobre una plataforma JAVA.
- **Personal**: el equipo de desarrollo está formado por el autor del presente documento y jefe del proyecto, Francisco Javier Paniagua Correa.
- **Especificaciones del proceso**: éstas quedan recogidas en el apartado 3.5 Descripción de requisitos.

7.1.2 Factores de escala

Los parámetros que contempla el modelo de estimación elegido con sus correspondientes valores son:

- **Precedencia**: considera el volumen de proyectos similares desarrollados tanto en aspectos organizacionales como en hardware y software. La Tabla 65 muestra los aspectos tenidos en cuenta para definir el valor del parámetro.

PRECEDENCIA			
CARACTERÍSTICAS	**MUY BAJO** **BAJO**	**NOMINAL** **ALTO**	**MUY ALTO** **EXTRA ALTO**
Entendimiento organizacional de los objetivos del producto	General	Considerable	Total ✔
Experiencia con software relacionado	Moderada ✔	Considerable	Amplia
Necesidad de innovación en el procesamiento de datos, arquitectura y algoritmos	Considerable	Alguna ✔	Mínima

Tabla 65: Factores de precedencia

De acuerdo a la información mostrada en la Tabla 65, el nivel de precedencia que se considera para este proyecto es **Nominal**.

- **Flexibilidad en el desarrollo**: es el nivel de exigencia en el cumplimiento de los requerimientos del sistema y plazos de entrega.

FLEXIBILIDAD EN EL DESARROLLO			
CARACTERÍSTICAS	**MUY BAJO** **BAJO**	**NOMINAL** **ALTO**	**MUY ALTO** **EXTRA ALTO**
Grado de cumplimiento de los requerimientos establecidos.	Total ✔	Considerable	Básica
Estímulo por la finalización antes de tiempo.	Elevado	Medio ✔	Bajo

Tabla 66: Flexibilidad en el desarrollo

Según la información mostrada en la Tabla 66, el grado de flexibilidad para este proyecto se considera **Bajo**.

- **Arquitectura/resolución de riesgo**: conocimiento de los ítems de riesgo crítico y el modo de abordarlos dentro del proyecto.

ARQUITECTURA / RESOLUCIÓN DEL RIESGO						
CARACTERÍSTICAS	**MUY BAJO**	**BAJO**	**NORMAL**	**ALTO**	**MUY ALTO**	**EXTRA**
Planificación de la administración de riesgo, identificando todos los ítems de riesgo y estableciendo hitos de control para su solución por medio de la revisión del diseño del producto (PDR).	Ninguna	Pequeña	Algo	General ✓	En gran medida	Completa
Cronograma, presupuesto e hitos internos especificados en el PDR, compatibles con el plan de administración de riesgo	Ninguno ✓	Pequeño	Algo	General	En gran medida	Completo
Porcentaje del cronograma dedicado a la definición de la arquitectura de acuerdo a los objetivos generales del producto	5	10	17	25 ✓	33	40
Porcentaje de arquitecturas de software disponibles para el proyecto	20	40	60 ✓	80	100	120
Herramientas disponibles para resolver ítems de riesgo, desarrollando y verificando las especulaciones de	Ninguna	Pocas	Algunas	Buenas ✓	Muy buenas	Completas

arquitecturas						
Nivel de incertidumbre en factores claves de la arquitectura: interfaz de usuario, hardware, tecnología, performance	Extremo	Significativo	Considerable	Medio	Poco	Muy poco ✔
Cantidad y grado de criticidad de ítems de riesgo	> 10 crítico	5 – 10 crítico	2 – 4 crítico	1 crítico ✔	> 5 no crítico	< 5 no crítico

Tabla 67: Arquitectura/resolución del riesgo

Según los datos aportados por la Tabla 67, el grado de resolución del riesgo ha sido considerado como **Alto**.

- **Cohesión del equipo**: tiene en cuenta las dificultades de sincronización entre los participantes del proyecto: usuarios, clientes, desarrolladores, encargados de mantenimiento, etc.

  Para este factor no se ha descrito su tabla de valores debido a que el equipo de desarrollo está formado, como ya se comentó antes, por el autor del documento, por lo que el grado de cohesión del equipo ha sido considerado como **Extremadamente Alto.**

- **Madurez del proceso**: representa el nivel de madurez estimada en relación al modelo de madurez del software. Se le asigna un valor **Nominal**.

Una vez que se han definido todos los parámetros que componen los factores de escala, se procede a introducirlos en el formulario correspondiente de la herramienta COCOMO. De esta forma, el resultado es el que se muestra en la Figura 72 que aparece a continuación:

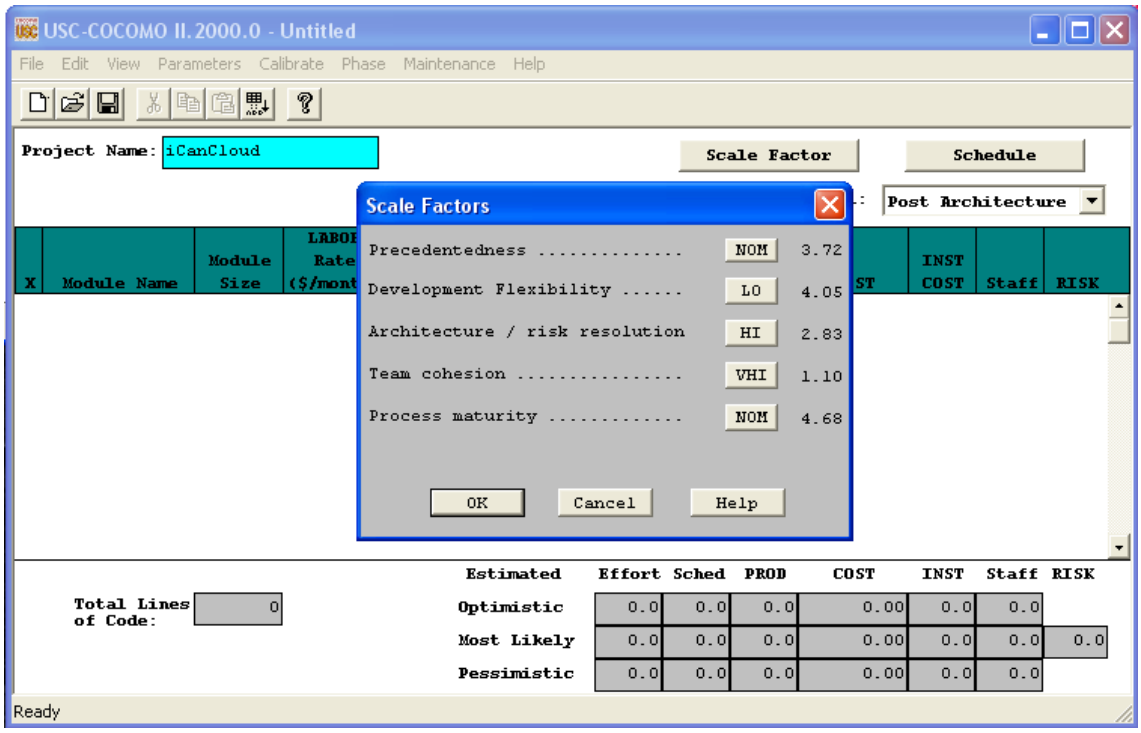

Figura 72: Factores de escala COCOMO

7.1.3 Multiplicadores de esfuerzo

En este apartado se van a tratar los factores que ajustarán el esfuerzo normal asociado al desarrollo con el fin de obtener una estimación más fiable. Para ello, se van a dividir dichos factores en cuatro áreas. A continuación se describirán cada una de ellas y se determinarán los valores estimados respecto a las mismas:

- **Producto**: restricciones y requerimientos del producto a desarrollar.
    - **RELY**: confiabilidad requerida del producto. Si el fallo del producto produce inconvenientes exclusivamente al desarrollador, el valor es bajo. Si, por el contrario, pone en peligro la vida humana, el valor es extremadamente alto.
    - **DATA**: tamaño de la base de datos.
    - **DOCU**: volumen de documentación acorde a las diferentes etapas del ciclo de vida.
    - **CPLX**: complejidad de las operaciones realizadas por el producto clasificadas en operaciones de control, computacionales, dependientes de los dispositivos, etc.

- **RUSE**: esfuerzo adicional necesario para que el software generado sea reutilizable.

PRODUCTO	
FACTOR	**VALOR ASIGNADO**
RELY	Nominal
DATA	Bajo
DOCU	Nominal
CPLX	Nominal
RUSE	Alto

Tabla 68: Tabla valores producto

- **Plataforma**: analizan la complejidad de la estructura hardware y software.
  - **PVOL**: volatilidad de la plataforma.
  - **STOR**: porcentaje de almacenamiento principal que usará el sistema.
  - **TIME**: restricción del tiempo de ejecución.

PLATAFORMA	
FACTOR	**VALOR ASIGNADO**
PVOL	Nominal
STOR	Nominal
TIME	Nominal

Tabla 69: Tabla valores plataforma

- **Personal**: miden el nivel de habilidad del equipo de desarrollo.
  - **ACAP**: capacidad del analista para el diseño, análisis, correcta comunicación y cooperación con el cliente y el equipo de desarrollo.
  - **APEX**: experiencia del equipo de desarrollo en aplicaciones similares.

- **PCAP**: habilidad del programador en la utilización de herramientas actuales y trabajo en equipo.
- **PLEX**: experiencia del equipo de desarrollo con la plataforma.
- **LTEXT**: experiencia en el lenguaje y las herramientas a emplear en el desarrollo.
- **PCON**: grado de permanencia del personal asociado a un proyecto.

PERSONAL	
FACTOR	**VALOR ASIGNADO**
ACAP	Alto
APEX	Nominal
PCAP	Nominal
PLEX	Bajo
LTEXT	Nominal
PCON	Alto

Tabla 70: Tabla valores personal

- **Proyecto**: condiciones y restricciones bajo las cuales se lleva a cabo el proyecto.
  - **TOOL**: mide el uso de herramientas software. El rango de valores va desde muy bajo, que implica el uso de herramientas sólo para codificación, edición y depuración, hasta extremadamente alto, que incluye herramientas especiales de integración del proceso de desarrollo.
  - **SITE**: mide el coste de la ubicación espacial y el soporte de las comunicaciones.
  - **SCED**: mide el grado de flexibilidad del cronograma exigido al equipo de desarrollo.

PROYECTO	
FACTOR	**VALOR ASIGNADO**
TOOL	Nominal
SITE	Bajo
SCED	Nominal

Tabla 71: Tabla valores proyecto

Llegados a este punto, en el que se han determinado los valores de los principales multiplicadores de esfuerzo del modelo de estimación elegido, se va a proceder a incluir dicha información en el formulario correspondiente de la herramienta COCOMO.

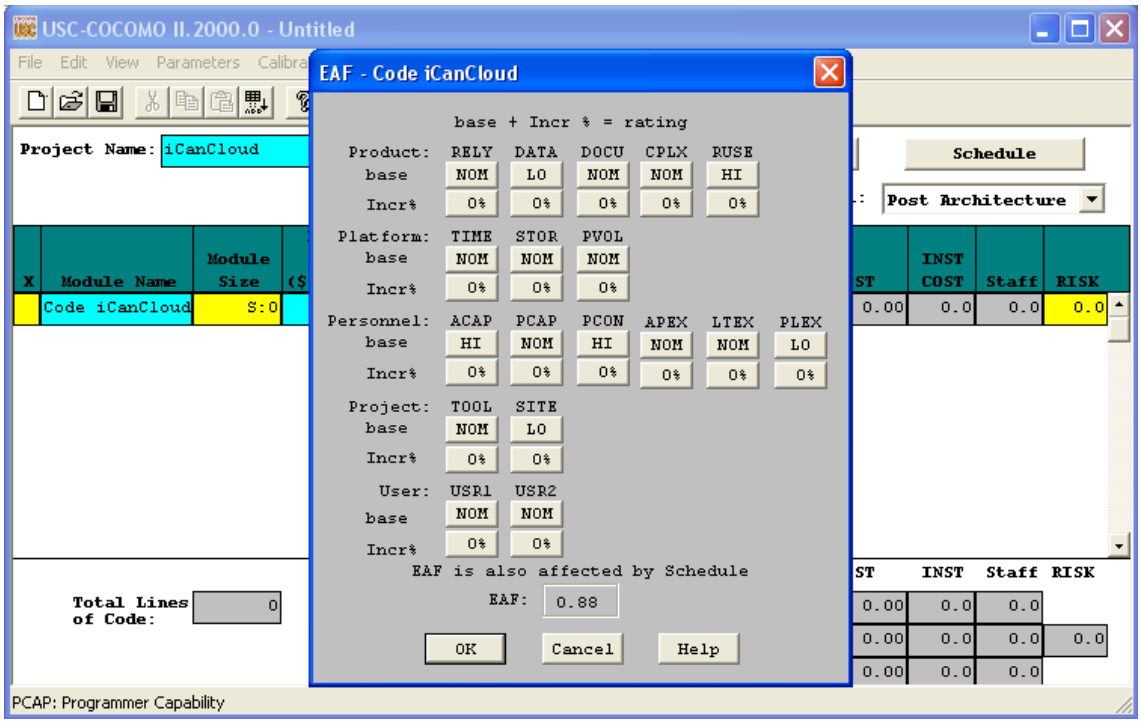

Figura 73: Multiplicadores de esfuerzo

La Figura 73 anterior muestra los datos recogidos durante este apartado en la herramienta de cálculo de costes. Como puede apreciarse, el factor de esfuerzo (**EAF**), para el conjunto de factores introducidos es igual a **0.88**.

7.1.4 Estimaciones

A continuación se llevará a cabo el análisis de los puntos función del proceso definido en el sistema. Éste posee la información especificada anteriormente sobre los modelos de estimación, factores de escala y multiplicadores de esfuerzo. Los puntos de función se calculan en base a los factores que se describen a continuación:

- **Inputs o EI (entradas externas)**. Entrada de datos del usuario o de control que ingresan desde el exterior del sistema para agregar y/o cambiar datos a un archivo lógico interno.

INPUTS	
CONCEPTO	**VALOR**
GESTIÓN DE MVS	4
GESTIÓN DE TAREAS	4
GESTIÓN DE CLOUDS	4
GESTION DE APLICACIONES	4
GESTIÓN DE SIMULACIONES	2
GESTIÓN DE RESULTADOS	2
GESTIÓN DE CONFIGURACIÓN	1

Tabla 72: Tabla estimación entradas

- **Ouputs o EO (salidas externas)**. Salida de datos de usuario o de control que deja el límite del sistema software.

OUTPUTS	
CONCEPTO	**VALOR**
GESTIÓN DE MVS	1
GESTIÓN DE TAREAS	1
GESTIÓN DE CLOUDS	1
GESTION DE APLICACIONES	1
GESTIÓN DE SIMULACIONES	2
GESTIÓN DE RESULTADOS	2
GESTIÓN DE CONFIGURACIÓN	1

Tabla 73: Tabla estimación salidas

- **Files o ILF (archivos lógicos internos)**. Cada grupo lógico de datos que es generado, usado o mantenido por el sistema software.

FILES	
CONCEPTO	**VALOR**
GESTIÓN DE MVS	1
GESTIÓN DE TAREAS	1
GESTIÓN DE CLOUDS	1
GESTION DE APLICACIONES	1

Tabla 74: Tabla estimación ficheros internos

- **Interfaces (archivos externos de interfaz)**. Archivos transferidos o compartidos entre sistemas software.

INTERFACES	
CONCEPTO	**VALOR**
GESTIÓN DE SIMULACIONES	9

Tabla 75: Tabla estimación interfaces

- **Inquiries (solicitudes externas)**. Solicitud de datos al sistema.

INQUIRIES	
CONCEPTO	**VALOR**
GESTIÓN DE MVS	2
GESTIÓN DE TAREAS	2
GESTIÓN DE CLOUDS	2
GESTION DE APLICACIONES	2

Tabla 76: Tabla estimación consultas

Una vez descritos todos los parámetros anteriores, se van a introducir todas las valoraciones en el formulario correspondiente. En la Figura 74 se muestra el cuadro de diálogo en el que se ha seleccionado el método de cálculo del tamaño 'puntos de función', el lenguaje de programación 'orientado a objetos' y se han introducido los valores obtenidos en las tablas anteriores.

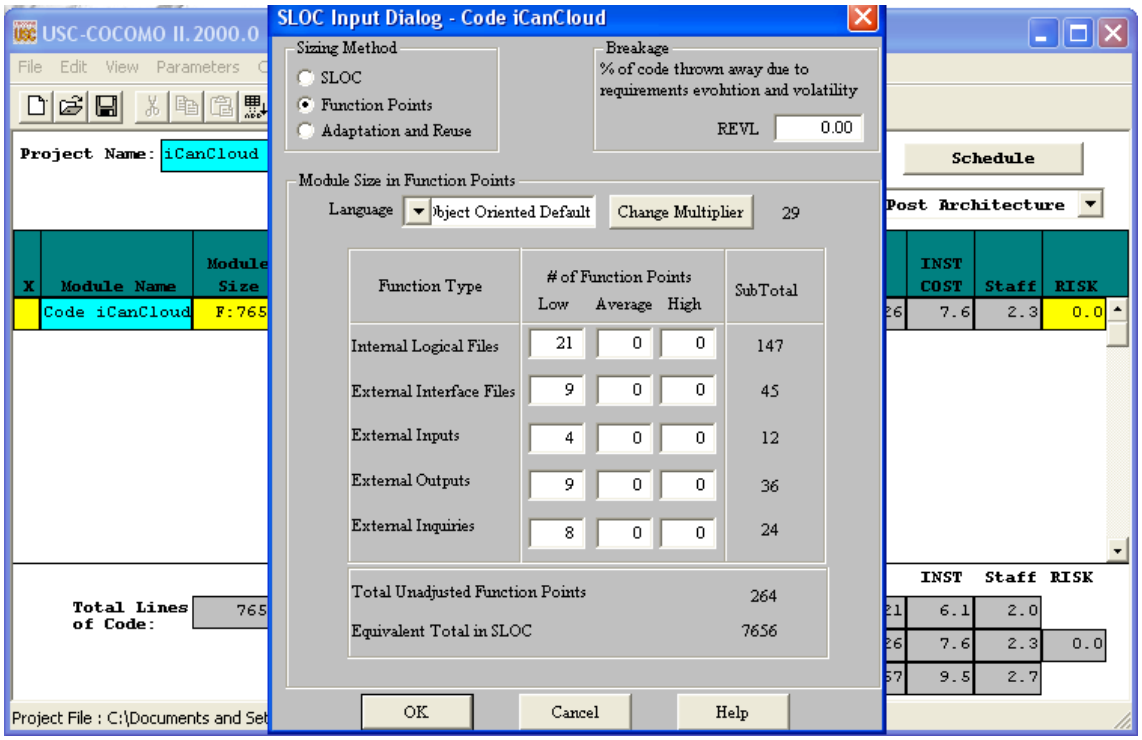

Figura 74: Puntos de función COCOMO

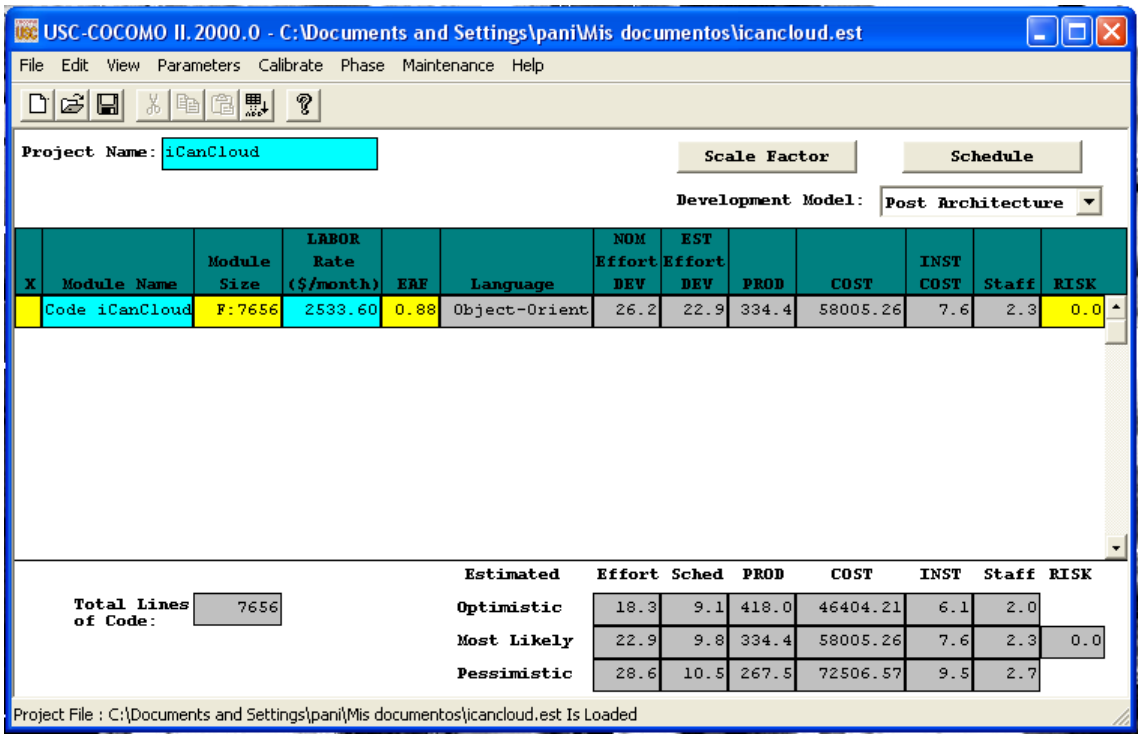

Figura 75: Resultados COCOMO

Como se puede observar en la Figura 74, tras introducir los datos asociados a las entradas, salidas, ficheros, etc., encontrados en el sistema, se obtienen un total de 264 puntos de función sin ajustar, lo que es equivalente a un promedio de unas **7656 líneas de código** de tamaño.

De esta forma, se han incluido ya en la aplicación todos los datos necesarios para realizar las estimaciones. Así, tras la introducción de los puntos de función, obtenemos unos resultados como los que aparecen representados en la Figura 75.

COCOMO						
Estimación	**Esfuerzo (Pers/Mes)**	**Duración**	**Productividad (SLOCs/per/mes)**	**Coste (€)**	**Coste por Instrucción**	**Personal necesario**
Optimista	18.3	9.1	418.0	46404.21	6.1	2.0
Realista	22.9	9.8	334.3	58005.26	7.6	2.3
Pesimista	28.6	10.5	267.5	72506.57	9.5	2.7

Tabla 77: Tabla información salida COCOMO

En la Tabla 77 puede observarse cómo la herramienta COCOMO proporciona una serie de estimaciones. Se obtienen tres estimaciones distintas: una optimista, otra pesimista y una última realista (teniendo así en cuenta un cierto factor de riesgo). Generalmente, es ésta última la que se tiene en cuenta en las predicciones. Por lo tanto, se estima que con una productividad media de **334 líneas de código por persona y por mes**, y en un plazo de **9.8 meses** se complete el desarrollo del proyecto, con un equipo de desarrollo formado por **2.3 personas**.

7.2 Cálculo del esfuerzo final

En este apartado se describe el esfuerzo requerido en la realización del proyecto descrito, incluyendo todos los gastos necesarios para su total elaboración y puesta en marcha.

- **Cálculo de horas totales**: en la Tabla 78 se muestran las distintas fases del ciclo de vida del proyecto, así como el número de horas necesarias para la realización de cada una de ellas.

TAREA	HORAS DEDICADAS
Análisis	20 h.
Diseño	30 h.
Implementación	350 h.
Pruebas	50 h.
Documentación	150 h.
Total	600 h.

Tabla 78: Desglose por actividades del proyecto

Como se puede apreciar en la tabla anterior, el esfuerzo total asociado al proyecto se eleva a las 600 horas de trabajo, donde más del 50% del mismo se ha dedicado a labores de implementación del código del sistema.

Con estos datos, si además tenemos cuenta la información que se recoge a continuación:

- La jornada laboral es de 4 horas diarias.
- Cada mes consta de 20 días laborales.

Llegamos a la conclusión de que el período de tiempo empleado en el desarrollo del proyecto es de aproximadamente 7,5 meses.

- **Salarios por categoría:** para la realización del proyecto se requiere personal informático cualificado, capaz de llevar a cabo las tareas reflejadas en la tabla anterior. Para ello será necesario que este personal adopte determinados roles, los cuales se recogen en la tabla que aparece a continuación.

CARGO	SUELDO NETO	SUELDO BRUTO	COSTE / HORA
Analista	1.532 €/mes	28.000 €/año	**25 €**
Diseñador	1.532 €/mes	28.000 €/año	**25 €**
Programador	1.225,6 €/mes	22.400 €/año	**20 €**
Responsable de Pruebas	1.225,6 €/mes	22.400 €/año	**20 €**
Responsable de Documentación	1.225,6 €/mes	22.400 €/año	**20 €**

Tabla 79: Salarios por categoría

En la Tabla 79 se recogen los salarios correspondientes a los distintos roles que el personal del proyecto deberá adoptar para la realización del mismo. En dicha tabla la información se divide en:

- **Coste / Hora**: indica el sueldo bruto en una hora de trabajo.
- **Sueldo Bruto**: indica el sueldo bruto anual, con 14 pagas mensuales.
- **Sueldo Neto**: indica el sueldo neto mensual. Se descuenta el I.R.P.F (17%) y Seguridad Social (6.4%)

- **Gastos de personal imputables al proyecto:** el personal encargado del desarrollo del proyecto, compuesto por un informático, ha necesitado adoptar todos y cada uno de los distintos roles especificados en la Tabla 79 para completar las actividades anunciadas anteriormente (Tabla 78).

  La siguiente tabla recoge el coste total de personal asociado al proyecto.

CARGO	PERSONAL	HORAS	COSTE /	TOTAL
Analista	1	20 h	**25 €**	**500 €**
Diseñador	1	30 h.	**25 €**	**750 €**
Programador	1	350 h.	**20 €**	**7000 €**
Responsable de Pruebas	1	50 h	**20 €**	**1000 €**
Responsable de	1	150 h.	**20 €**	**3000 €**

TOTAL		600 h.		**12250 €**

Tabla 80: Gastos de personal imputables al proyecto

De acuerdo a la información mostrada en la Tabla 80, los gastos asociados al personal encargado del desarrollo del proyecto ascienden a **12250 €**.

- **Recursos materiales empleados:** la Tabla 81 muestra el coste de los recursos necesarios para la realización del proyecto. Estos costes ya incluyen el impuesto del I.V.A.

RECURSO	CANTIDAD	COSTE TOTAL
Ordenador personal	2	**1200 €**
Router	1	**90 €**
Cable RJ 45	2	**30 €**
Microsoft Office XP	1	**199 €**
Microsoft Windows XP	1	**499 €**
Ubuntu	1	**0 €**
Simulador iCanCloud	1	**0 €**
TOTAL		**2018 €**

Tabla 81: Recursos materiales empleados

Como se puede apreciar en la tabla anterior, el simulador de cloud computing "iCanCloud" es de código libre, por lo que el coste al proyecto de una distribución es de 0 euros. Además, este simulador requiere de un sistema operativo basado en Unix para su funcionamiento. Se ha optado en este caso por una distribución de Ubuntu, la cual proporciona todas las herramientas necesarias para cumplir los objetivos del proyecto y además su distribución es gratuita.

- **Gastos indirectos:** la Tabla 82 muestra los gastos indirectos que repercutieron en los costes totales para el desarrollo del proyecto.

DESCRIPCIÓN	COSTE
Alquiler del local	Incluido en los gastos indirectos
Electricidad	Incluido en los gastos indirectos
Agua	Incluido en los gastos indirectos
Productos de limpieza	Incluido en los gastos indirectos
Amortización Inmobiliario	Incluido en los gastos indirectos
Gastos de Comunidad	Incluido en los gastos indirectos
Costes de estructura	Incluido en los gastos indirectos
COSTES INDIRECTOS (10%)	**12250 * 0,10 = 1225 €**

Tabla 82: Gastos indirectos

- **Resumen del presupuesto:** la Tabla 82Tabla 83 muestra un resumen de todos los costes que ha requerido el proyecto, así como la suma total de los mismos.

DESCRIPCIÓN	COSTE
Personal con cargo al proyecto	12250 €
Recursos materiales empleados	2018 €
Gastos indirectos	1225 €
TOTAL	**15493 €**

Tabla 83: Resumen del presupuesto

Al coste total calculado hay que añadir un margen de imprevistos del 10%: **1549,3 €**

Coste total + margen de imprevistos: 15493 € + 1549,3 € = **17042,3 €**

Finalmente se calculan los beneficios a obtener con el proyecto, un 15% del coste total: **2556,34 €**

Coste total + margen de imprevistos + beneficios = 17042,3 € + 2556,34 € = **19598,64 €**

El presupuesto total del proyecto realizado es de **19598,64 €, I.V.A. incluido.**

Anexo

2

8. Anexo II – Bibliografía y referencias web

[1]. Lea Douglas. “Programación concurrente en Java: principios y patrones de diseño”. Addison-Wesley. 2001.

[2]. Akif, Mohammad. “Java y XML: referencia para programadores”. ANAYA Multimedia. 2002.

[3]. Eichelberger, H. “A comprehensive analysis of UML tools, their capabilities and their compliance”. University of Hildesheim, Institut für Informatik, Software Systems Engineering. 2009

[4]. Bennet, Simon. “Análisis y diseño en sistemas orientados a objetos usando UML”. McGraw-Hill. 2007.

[5]. A. Nuñez, J. L. Vázquez-Poletti, A. C. Caminero, J. Carretero, I. M. Llorente. “iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator”. Departamento de Informática, Universidad Carlos III de Madrid.2011

[6]. Buyya, R., Murshed, M. “GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing”. Concurrency & Computation: Prac. & Exp. 14. 2002

[7]. Buyya, R., Ranjan, R., Calheiros, R.N. “Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities.” In: Proc. Of the 7th High Performance Computing and Simulation Conference (HPCS). 2009

[8]. Foster, I., Zhao, Y., Raicu, I., Lu, S. “Cloud Computing and Grid Computing 360-Degree Compared”. In: Proc. Grid Computing Environments Workshop. 2008

[9]. Boehm, Barry W. “Software cost estimation with COCOMO II”. Prentice-Hall. 2000

[10]. Gil Corrales, Susana. “Creación de un curso de autoenseñanza para Internet sobre la herramienta USC-COCOMO II 2000 utilizando una factoría de cursos”. S. Gil. 2002

Anexo 3

9. Anexo III – Glosario de acrónimos y abreviaturas

En la Tabla 84 que se muestra a continuación, aparece la relación de los principales acrónimos y abreviaturas que aparecen a lo largo de este documento con sus significados correspondientes.

Acrónimo	Descripción
App	Application o aplicación
ARP	Address Resolution Protocol
CC-001	Componente Controlador 001
CM-001	Componente Modelo 001
COCOMO	Modelo Constructivo de Costes
CPU	Unidad central de procesamiento
CV-001	Componente Vista 001
GUI	Graphic User Interface o Interfaz Grafica de Usuario
IP	Internet Protocol
Java 2 EE	Java 2 Enterprise Edition (antes conocido como Java 2 Platform, Enterprise Edition o J2EE hasta la versión 1.4).
JRE	Java Runtime Environment o entorno en tiempo de ejecución de Java.

MAC	Media Access Control o control de acceso al medio.
MIPS	Millones de instrucciones por segundo
MV	Maquina virtual
MVC	Modelo Vista Controlador
PDF	Portable Document Format
RAM	Random-Access Memory o memoria de acceso aleatorio.
SSH	Secure Shell o intérprete de órdenes segura.
TCP	Transmission Control Protocol
XML	eXtensible Markup Language

Tabla 84: Acrónimos y abreviaturas